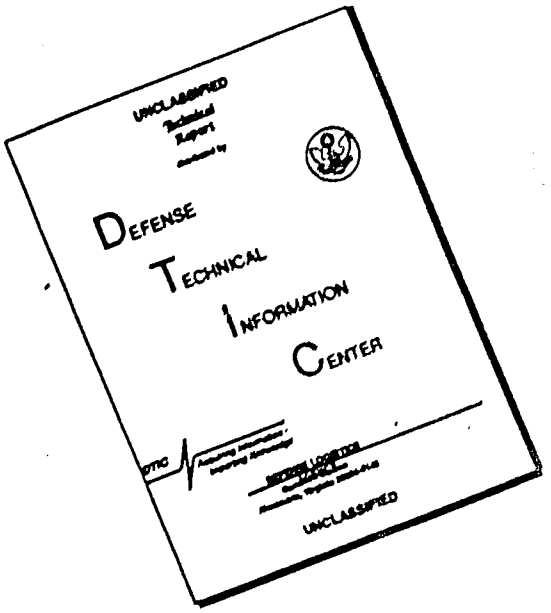
DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

MEMORANDUM
RM-3324-PR
NOVEMBER 1962

THE FIFTH RAND COMPUTER SYMPOSIUM

F. J. Gruenberger, Editor

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-3324-PR
NOVEMBER 1962

THE FIFTH RAND COMPUTER SYMPOSIUM

F. J. Gruenberger, Editor

This research is sponsored by the United States Air Force under Project RAND — Contract No. AF 49(638)-700 monitored by the Directorate of Development Planning, Deputy Chief of Staff, Research and Technology, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force. Permission to quote from or reproduce portions of this Memorandum must be obtained from The RAND Corporation.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA

PREFACE

This Memorandum is an expurgated transcript of the Fifth Annual RAND Computer Symposium, held at The RAND Corporation, April 1962. The idea for these symposia grew out of the observation that much of the value of computing industry conferences comes from the informal conversations which take place in hotel rooms and conference corridors. The feeling that an organized "bull session" might be a worthwhile endeavor, led to the invitation of some twenty individuals to come to RAND for a full day of discussion on common problems in the computer field.

These sessions have been held annually since 1958 on the day just prior to the Western Joint Computer Conference. The Symposium is, in effect, a meeting of individuals prominent in the industry. The views expressed in this transcript are those of the individuals involved and not those of their employers nor of The RAND Corporation.

The discussion during the fifth symposium centered around the topic, "Pros and Cons of Common Languages," with special consideration being given to use by the military of Command and Control languages. Because this topic is of special interest to the U.S. Air Force, and the computing field in general, the transcript of the 1962 Symposium is being released as a RAND Memorandum.

An article based on this transcript appears in the October and November 1962 issues of Datamation magazine.

SUMMARY

The Fifth Anual RAND Computer Symposium, held in Santa Monica on April 30, 1962, was concerned with the single topic, "Pros and Cons of Common Languages."

RAND was represented by Paul Armer, George Armerding, Fred Gruenberger, Jack Little, and consultant Robert L. Patrick, who chaired the session. Other attendees were:

Phillip Bagley, MITRE Corporation

Howard Bromberg, RCA

Tom Cheatham, Computing Associates

Richard Clippinger, Minneapolis Honeywell

Joe Cunningham, U.S. Air Force

Bill Dobrusky, System Development Corporation

Bernie Galler, University of Michigan

Barry Gordon, IBM

Jerry Koory, System Development Corporation

Brad MacKenzie, Burroughs Corporation

Dan McCracken, McCracken Associates

Ascher Opler, Computer Usage Company

Charles Phillips, Business Equipment Mfrs. Assn.

Dick Talmadge, IBM

The seven-hour discussion dealt with the advantages and disadvantages of using higher level languages for instructing computers; the feasibility of making one of these languages a standard or common language; and the applicability of these languages (e.g., FORTRAN, COBOL, ALGOL, JOVIAL,

(NELIAC) to Command and Control problems.

The original transcript was edited and corrected by each of the attendees. This Memorandum contains the final version of the transcript.

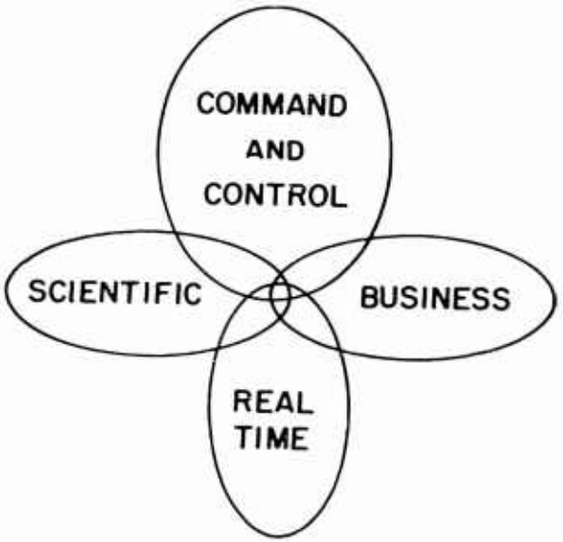
THE FIFTH RAND COMPUTER SYMPOSIUM

CHAIRMAN PATRICK: Our objective today is enlightenment. The subject of common languages is clouded by claims in advertising that are unsupported. Some examples are hung on the walls here today.

I have here two bags. This large paper bag is for opinions. This small chamois bag is for facts. From time to time during the session, a speaker may not make his meaning clear. Should this occur, I will exercise the chair's prerogative and shall classify his speech by displaying the appropriate bag. If I should happen to misclassify someone's speech, the speaker should attempt to fill us in to convince us that his remarks are not opinions but are indeed facts

(because that's probably the way I will misclassify). Please supply the supplementary information before you yield the floor.

Let's look at this chart that we have over here. (See Figure 1.) This chart will indicate why we should all be concerned. It is rumored that the Department of Defense (DOD) intends to standardize on a language for Command and Control. On the right the chart shows an area characterized by the name, Business, usually characterized by a high ratio of input and output to computing. On the left we also have Scientific jobs which are characterized by just the inverse. We have all heard of the famous Los Alamos job in which you read one card in the morning, compute all day, and print one line in the afternoon. Of course, this job doesn't exist; in fact neither of these areas exist in pure form, but sections and subsections of jobs do fall into these categories. There is an overlap of the two as indicated by this Venn diagram intersect. This overlap is seen in areas like data reduction and heavy actuarial work. In these areas you may think of it as scientific, but it has many characteristics of a business job. Command and Control has characteristics which are common to these two and also many characteristics of its own. Real-time applications have characteristics common to Command and Control and also characteristics unique to real-time problems. Consequently, if anything were to happen such as a standardized language for Command and Control, it would affect the entire field.



COST = $TRAIN + $PGM + $CODE + $COMP + $ASSEM + $TEST + $PROD + $MAINT + $DOC

Figure 1

The lower portion of the chart shows a simple equation. The equation is an expression for total cost. (A similar expression could be written for time.) As the equation shows, total cost is made up of dollar amounts spent for training, programming, coding, compilation, assembly, testing, production, maintenance, and documentation. All of these factors are involved in every single job that is done on a computer. (In some cases the coefficient of a given term may be zero. For example, if you have the facility for compile-and-go, then the assembly phase does not appear.) But for most problems, all the terms exist with non-trivial coefficients.

Sales literature and some of the "technical" literature that we read implies that we can reduce some one of these terms without at the same time increasing some of the other terms. The gullible citizens believe these statements. I believe that we will be doing ourselves and our country serious damage if we fail to see through these claims. During the discussion today, I invite you to use this chart to explain what you mean.

Fred Gruenberger has another chart which he would like to show you. Fred, why don't you take them around the carrousel?

GRUENBERGER: I hope you'll forgive us for starting out somewhat formally this morning. We have had quite a few meetings, both formal and informal, in the last few weeks, discussing this problem. We have noticed that during such discussions, people tend to go around and around a merry-go-

round in talking about common languages, so we made up this chart. It's really just a sort of checklist of the topics that keep coming up over and over (see Figure 2).

Let me just go around these items very quickly. The first thing that we have found that causes confusion in a discussion of common languages is the tendency for people to interchange, sometimes in the middle of a sentence, the ideas of "what can a higher level language do for us?" (For example, what goodies do we get by using ALGOL?) and "what advantages do we gain by using a common language?" Common languages are those which cut across installations, problems, and machines.

The other topics are pretty obvious. How are you going to train people? What kinds of people are going to use a common language? (We're talking here about the level of the users. Are they going to be top-notch programmers, or are they going to be clerks?) Can the system be changed by the masters and can it be changed out at each installation where it is used? Do we lose efficiency due to commonness? This is a claim that has been made, although it might be refuted. In other words, if we implement a given language on a given machine, have we taken advantage of some of the characteristics of that machine that might be good?

Does the language exhibit design control? That is, was it designed by a competent man who retained control over its ingredients or was it designed by a committee? Does the system freeze progress? Is it going to be maintained? Here

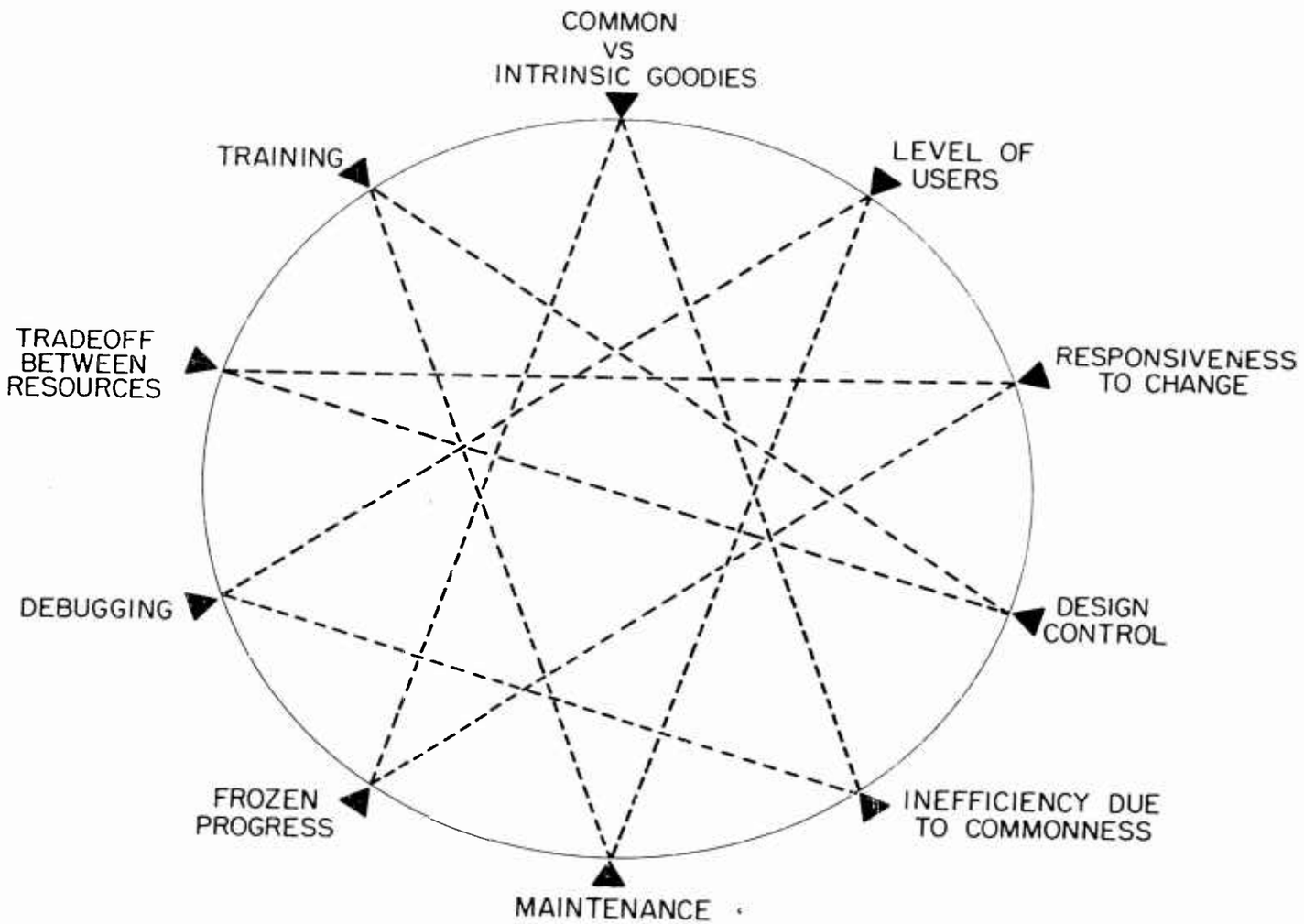

COMMON
VS
INTRINSIC GOODIES
LEVEL OF
USERS
RESPONSIVENESS
TO CHANGE
DESIGN
CONTROL
INEFFICIENCY DUE
TO COMMONNESS
MAINTENANCE
FROZEN
PROGRESS
DEBUGGING
TRADEOFF
BETWEEN
RESOURCES
TRAINING

Figure 2

again, we have the global and local problem. Can and will it be maintained globally and can it and will it be maintained locally? How will you debug in this language? Will the language and its system allow local managers to trade off their resources; namely, people and machine time?

We have noticed that as people discuss this subject, they bounce from one item to another on this list. We don't show all the forty-five connecting lines, but they should be here.

PATRICK: Now would anyone like to open up the informal discussion with a fact?

OPLER: I would rather open with a question. Do you distinguish between a common language and a standard language?

PATRICK: I think we would like to if you would care to describe them for us.

OPLER: I'm not sure I can, but perhaps I could give examples. I suppose there is really a Venn diagram intersection between the two. A common language is one that would be acceptable to two or more disparate machines. A standard language is one maintained and enforced as a standard language which requires some sort of committee agreement.

Some manufacturers had developed languages which could be processed on several of their machines. These would be common languages, but they would not be standard languages.

LITTLE: Are we saying that a common language may not be the standard, but a standard language, by definition, must be common?

GALLER: Could I ask whether there is the feeling here that FORTRAN is a common language? You can say, of course, that there is no one FORTRAN but this isn't very important. Could we use FORTRAN as an example of a common language?

McCRACKEN: It's the closest thing there is.

GRUENBERGER: Yes, close, but it doesn't make it. I know of no example of a language that is common in the full sense of the word, but FORTRAN does come close.

McCRACKEN: I think there is a principle involved here which is going to operate on us all day. It goes like this. This is what we've got; it will do our work for us. Someone comes along and proposes something better; perhaps a standard. And everyone then says, "Gee, it isn't perfect and, therefore, it's no good." The question is whether the things that are proposed are better than what we have now, not how they match up against some ideal.

BROMBERG: Going back to Opler's attempted definition, you could conclude (just as we concluded before) that there is no such thing, really, as a common language; that there is no such thing, Dan, as a standard language.

If we consider that, in order to handle a standard language, it is necessary to have an authoritarian group to establish and maintain it, then you aren't going to have a standard language. If it is necessary that all ten of the things listed on that chart be applied to the language by one central group, then this can exist only in a finite world

such as the world of a given manufacturer or a single installation. In that case someone in this finite world can take a common language and define it to be their standard. But as far as a national standard language goes, there is no such thing.

But I feel it is unnecessary to labor this distinction between common and standard. For our purposes here, it seems to me we can discuss only common languages, by which we mean those which can function on more than one machine type.

GRUENBERGER: I didn't say it had to be perfect, Dan. I simply said that we don't have one.

PATRICK: Maybe it would be appropriate to discuss whether we want one. I think there are some of us who do, but then there are also some of us who don't. Dick, would you like to discuss this point?

TALMADGE: Whether we want a common language or a standard language?

PATRICK: Let's consider them synonymous for a while.

TALMADGE: It's my opinion that we don't want a common language in the sense that most people use that term; that is, a language that everybody would use. To draw an analogy, mathematics, which people tend to think of as using a standard language, in reality uses a standard set of notations that are widely understood. Furthermore, a mathematician may use his own notation within very elastic limits, as long as the meaning is clear. One of the most acute problems in computing

is that we have not yet developed procedures which allow such a standard notation. Until we know more about linguistic structure, it is unlikely that we will. In any event, a common language (whatever that may be) does not seem to be the answer.

PATRICK: So you say we're not quite ready yet. Perhaps we have something more to learn.

TALMADGE: I think we have a lot to learn about how to use a language to express a given class of problems; and how to determine what problems are amenable to a given language.

BROMBERG: It strikes me that the same comment can be made with reference to every single human endeavor; we have a lot to learn. Name anything--science, living, or what have you--we have a lot to learn. But this in itself should not preclude our being able to use that which we have or that which we could have. There still exists a long list of jobs that have to be done with perhaps less than perfect tools. If something better comes along, why not use it?

TALMADGE: I didn't intend to imply that one should not use whatever tools are available. I merely meant that I think the present tools are inadequate for the job of forming a permanent language; rather, they should be used to develop more powerful tools.

BROMBERG: I assumed that the question was, "Is this the time for a common language?" and I would say this is certainly the time.

GORDON: I would like to point out, Howard, that there is a significant difference between utilizing and standardizing. I would also like to quote from a letter written by Dick Hill...

PATRICK: Is that Richard H. Hill of Informatics?

GORDON: Richard H. Hill, then of Western Data Processing Center. This was a letter written in June of 1960 which I clipped out of a SHARE secretary's distribution. He has something to say about a language system (well known by its initials), and I quote,

> One of the worst curses of mankind is premature standardization. Reasonable standards evolve; they are not imposed. The penalties of premature standardization have proved to be far more costly than the rewards.

I think everyone is in favor of standardization in much the same way everyone is in favor of motherhood; but also, like motherhood, if it occurs prematurely, it can cause a great deal of inconvenience.

GALLER: I'd like to call attention to an example of successful standardization that came along at a certain time and was successful for a certain reason. I'd like to hear a discussion some time of why it was successful and why some others weren't. I'm referring, of course, to SAP. At the time SAP was made a standard of SHARE, it served a very useful purpose. I think it contributed greatly to the progress we have made from that time on. At that time, there was a need for something and SAP filled that need. I can't explain what's so different about things now except possibly that we know more now. We are perhaps a lot more individualistic

today, and we can evaluate things better.

It's to everyone's advantage to be able to communicate. The more standardization we have, the more we can communicate easily. But there is a cost to standardization. Each person must ask himself, for example, "Should I use FORTRAN-IV? What is the cost of deviating from using this language? Likewise, what does it cost other people if I deviate or if they do or do not deviate?" I don't think there's a clear case for either side.

ARMERDING: The problems involved in standardizing on SAP were several orders of magnitude smaller than the problems that we have in standardizing on these magic languages today. Basically the problem then was to get a group of guys together to standardize on what three-letter mnemonics we would assign to each of the machine op-codes plus a few pseudo-ops to get the assembly system running. All of that amounts to a trivial problem compared to standardization in one of the non-machine languages today. The difference is so great that I don't think we can really compare them.

GORDON: Point of information. If we're going to talk about standard languages and we call SAP a standard language, we had better clarify what we mean. I don't think SAP was a standard; it was one of a great many languages for that class of machines.

BROMBERG: I wonder if we couldn't outlaw the use of the word standard at least for the time being.

GORDON: Either that or define it.

BROMBERG: Let me give you a little of the background of what is going on in ASA, BEMA, and the X-3.4 Subcommittee on Common Programming Langugaes. There has been established a survey director whose function it is to survey all the common programming languages. Remember, these are languages which have been developed to be used on more than one machine type. A preliminary survey which reflects languages of broad utility has been done, and it has been sent to various overseas contributors. An interesting part of this survey is that assembly languages like SAP, Autocoder, and X-1 have by definition been precluded. They are not incorporated in the survey because of obsolescence or the level of the language. The only kind of common language admitted to the survey is that which is effectively non-machine oriented; that is, it does not look like any known machine order code. The entries are essentially problem languages as close to natural language as we can get.

McCRACKEN: How many common languages have you listed?

BROMBERG: There must have been about 90. I can't be too sure.

PATRICK: Howard, I can't think of the <u>first</u> one, if you set up the requirement that it must be machine independent.

BROMBERG: Of course, it all depends on what you mean by machine independent. Must it be totally machine independent?

PATRICK: Well, if you are going to cut across machine lines it seems like it ought to be.

McCRACKEN: It doesn't have to be perfect in order to be better than what we now have. That's going to be my standard speech today.

GRUENBERGER: Why don't you give that speech the code name "Monkeywrench" then?

GORDON: But the new language we adopt should be significantly better than what we now have.

GALLER: I'd like to point out that SAP is probably a small isolated example of what we're talking about. But at the time it was the language. At that time the decision to standardize on it was a major decision. As a communication language it had some of the elements of what we are talking about now.

GORDON: Not at all.

GRUENBERGER: For one thing, it dealt with only one machine type. You could just as well say that 650 machine language is a common language among 650 users. They can't avoid it.

ARMER: Maybe that's why we're having so much trouble now. We don't have enough tandardization at the trivial end. For example, on character sets.

PATRICK: I seem to hear three different cheers for that sentiment.

OPLER: I'd like to say one more thing about the SAP question before we lay it to rest. We must remember that at the time of development of SAP there were three situations

which do not apply today. Firstly, everyone was bleeding from 701 experience where there was no standardization effort. Secondly, SAP was pretty well done before the 704's started to arrive and before there had been a heavy investment in 704 programming. And, thirdly, there were two potential rivals already in evidence--NYAP and SAP.

In the context of those three items, I'm inclined to agree that the decision to standardize on SAP was a good one. I also agree that it is not too pertinent to the situation we have today.

LITTLE: One wonders why this was such a big step forward and why we took such a big step backward with the advent of the 7090. We seem to have diverged again.

Languages may evolve or standards may evolve. I think that languages can get to be standard or common in a lot less exotic ways. If a big enough user uses a single language without regard to the rest of the world, then you are sort of saddled with that language. DOD may be going down this road and whether you like it or don't like it you may have the situation with you very shortly. I would like to hear an outline of some methods that can be used to help insure that the language we get is a good one. I am sure that we are going to get one. But I don't think we're going to get one that somehow magically everyone will like.

There are certain things that you want built into it. For example, you want it to have the ability to change--to

make itself better, or for people to make it better. There are two items on our chart that seem to deal with this question. One has to do with the language itself and another one has to do with the problems that we are going to use the language on. You want the language itself easy to change but if the language does not also allow you readily to change the problem that the language is used on, then it's going to fail.

So I'd like to see some discussion here of practical ways which will insure that the language or languages that do evolve will have in them desirable characteristics.

PHILLIPS: I'd like to go back and add something to what Howard said about the ASA-X3 program. One part of that program concerns the development of a standard language. I'm quite sure (although this may be just an opinion) that the 18 members of BEMA or the general interest and user groups that are supporting this program would not feel that they are bound to adopt and use, to the exclusion of everything else, a language that might be developed as a result of the program. By the same token, I don't think that Defense, in supporting the COBOL program, had any intention of making it a required program for use throughout Defense unless there was good reason for it; that is, unless it would serve a useful purpose and there were definite advantages or needs for communication that it could satisfy.

PATRICK: Well, that sounds very high and lofty but it seems to me that if you insist on COBOL before you will

consider a machine from a given manufacturer that this kind of implies that you won't get COMTRAN.

GORDON: Or FACT.

PATRICK: Well, if we're looking to the future it just doesn't seem that with the money press that is coming on the manufacturers that they can afford to do two of these languages. So if the DOD says, "I want COBOL," or "I want JAZZY," or some other darn language, it kind of looks like we're all going to be using it whether we like it or not. I would dearly like someone to convince me that I'm wrong.

LITTLE: Isn't it true that if you just get enough people implementing COBOL, either you accept it at that point of time as common or as a standard, or what you develop thereafter as common or standard must be compatible with COBOL?

Mathematics may be a standard language to write things in but you don't have a lot of people sitting around rewriting mathematics. They may disagree in the notations but they don't have to go back and rewrite totally. But when you have a program or a whole set of programs running on a machine and a new language comes along it implies a tremendous amount of work on the part of people to conform.

TALMADGE: I'd like to amplify my previous remarks about mathematical notations. In trying to standardize without enough knowledge, we may entirely miss the goal. For example, three or four people here have used the word "language" to categorize systems I would classify on completely different

levels: SAP and COBOL for instance are not even comparable. Perhaps we should first define the meaning of the much abused word "language."

MacKENZIE: Good, bad, or indifferent, I think we ought to be able to rigorously define the language in question.

PATRICK: A la the syntactical chart?

MacKENZIE: Not at all, for the charts are merely visual aids. We should be able to find ways of producing rigorous descriptions of languages--ones that people can read and ones that represent the authority rather than ones which require inferences to the authority based on observation of a machine representation in action.

McCRACKEN: I would tend to agree with that.

Suppose you have two languages that do about the same thing. One of them has a lot of effort behind it; that is, a lot of people working on it. The other is what you might call an offshoot of the same thing but just a bit better--but not widely used, and not being heavily worked on by large numbers of people. It's my opinion that the computing world would be better off to settle on the one that is more widely accepted and work within the framework of that one language to improve it, than to push for acceptance of offshoots which may, in some way, be better.

PATRICK: This is contrary to the basic assumption of all this standards jazz. In the speeches that Bright and Co. made on the East Coast last fall, they started out by assuming

that you couldn't do anything if it was just one manufacturer's work. A situation which could be described as "I'll assume FORTRAN out of the way and then I'll look for a common language."

LITTLE: Dan, did you say work on or work with? You said work on the language. I assume you mean a lot of people are using the language.

McCRACKEN: Well, I actually meant a lot of people fooling with compilers.

CLIPPINGER: You mentioned throwing out FORTRAN; perhaps I missed something before I came in. Has someone brought up the fact that X-3.4 has selected three languages to consider for standardization, of which one is FORTRAN?

PATRICK: No, I hadn't heard that.

GRUENBERGER: And what are the other two?

CLIPPINGER: COBOL and ALGOL. I assumed that everyone knew that. X-3.4 has requested from IBM a statement as to their position on this matter and IBM has responded favorably, and IBM has done some work in providing as a starting point an initial draft of a form of FORTRAN which could be used as a kick-off. X-3.4 is about to set up a group to go to work on FORTRAN along with the other two. You can put that in the fact bag.

PHILLIPS: Dick Clippinger was on the same panel you mentioned that discussed this subject back at the Eastern

Joint Computer Conference. If I'm not mistaken you have quoted Herb Bright wrong. I don't believe he said that because FORTRAN was a one-company language it should be outlawed. In fact, I think he generally supported FORTRAN as a candidate for a standard language.

CLIPPINGER: That's right. At that time he was very much distressed that X-3.4 was considering ALGOL and COBOL as standard languages and not FORTRAN.

PATRICK: Well, I didn't hear his speech but I have a copy here in front of me. I hope I'm not quoting him out of context but he says, "Unlike ALGOL and COBOL, it (FORTRAN) was not developed by a broadbase comparative study group but by a single manufacturer." He doesn't seem to mention the manufacturer's name. And after that he doesn't seem to mention FORTRAN at all. He talks a great deal about ALGOL and COBOL which looks like some sort of an indictment.

CLIPPINGER: Might you not have read in the indictment in your own mind?

PATRICK: I guess it looks that way.

PHILLIPS: I'm sure you must have, because in the talks we had before the conference his ideas were the reverse of what you imply. He was saying that just because it was developed by one company is no reason to throw it out.

ARMER: But it was a fact that at that time FORTRAN was not being considered by the committee.

GALLER: ALGOL and FORTRAN are quite similar in that they both cover what we have loosely called the scientific area. You really don't need both of them. So if there's a move to make them both standard, it would be interesting to see why. And here it seems to me that FORTRAN is being looked upon as a standard, not because it's so wonderful but because it is already so common. ALGOL, on the other hand, doesn't have much claim to commonness yet but it does have something to offer. I don't think there is a real tremendous need for two languages to be standard but it is interesting that both of them are being looked at.

McCRACKEN: But then why look at three of them? I'm not referring to COBOL but, for example, why do we also need MAD?

GALLER: MAD is a language that is not used very much yet. It seems to me that we cannot decide that any one of the languages we have now is it.

GRUENBERGER: You'd better use a different word, Bernie. You meant a lower case "it" didn't you?

GALLER: Sure, but look back to when FORTRANSIT was distributed. The covering letter for it said something like, "Everyone recognizes that FORTRAN is the language that we're going to use from here on out so you'd better get on the bandwagon." Simply and historically we know that that was not true. I don't even want it to be true. There's a difference between standardization at a point in time where you could get great benefits from it, and abandoning the search

for something better. We should always be prepared to go on to something better. We may choose to develop a language that is different, and the fact that various people are using MAD would seem to indicate that there is something there. There are people who are choosing not to take it on because of the cost of deviating from FORTRAN and ALGOL and so on. We are very happy that people are looking at it, but some people who have looked at it have rejected it, and that's fine. There is always a cost involved in deviating. We have examined that cost. We are examining it right now. We are asking ourselves whether it is worth while to rewrite MAD for some other machine or should we try to make a new version of it. We ask ourselves what is the cost involved to us. We would have to make some attempt to cover ourselves and to be able to translate from the old to the new. If this cost is less than the cost of deviating then we'll go ahead. We have got to be free to make changes if only to allow ourselves to keep progressing. People tell me, "You don't have tremendous production problems." This is fairly true. We do a lot of work but the life span of the jobs we do is pretty short. That means we can write off the life of these jobs to a large extent. There is a need for people to keep looking at these languages and to keep experimenting. And I think it is a mistake to throttle these efforts by decree or by economic pressures or what have you.

* : Who said we should do that?

---

* = Unknown voice.

GALLER: All right, let's say by talks at conferences which say we should depend on the manufacturers and we should stop doing things because it's time to standardize. (I think that's what Opler once said.) We've got to look at where we're going and each person has to look at each of these languages.

McCRACKEN: The question was whether in the process of looking at the new languages we have to create a new name at the same time. Can't we work within the framework of what is widely accepted? Can't we try to improve what we already have or do we have to go off to the side and create something new that is not compatible?

GALLER: Sometimes to make progress we have to go off to the side and start fresh.

PATRICK: Bernie is making use of the unique position he is in, where he can experiment without severely changing the course of the field unless he just happens to find something very good. Perhaps IBM cannot do this and the Department of Defense cannot do this. If either IBM or the DOD (and they are the two powers) do this, their experiments will have a profound effect on the field.

ARMER: God help us if our universities can't experiment.

McCRACKEN: That isn't what I said. They've got to experiment.

ARMER: What would you have them do differently? Somehow I feel that you're picking on Bernie.

McCRACKEN: No. Bernie is a friend. But if they do find something good by experimenting in the universities what do they do about it? Do they go out and say, "Let's everybody buy my language now? It just happens to be called MAD and it's not ALGOL." Or should they get back into the ALGOL effort, having found a good thing? Of course they should experiment.

ARMER: But it seems to me that their experiment has to involve a fairly large number of users. They can't just develop it and then not use it.

GALLER: We've got 2,000 people on campus just writing programs. We use it.

ARMER: Yes, but it seems to me Dan was arguing that MAD is not really going to change the world and now you should put it on the shelf and do something in ALGOL. Is that what you were saying Dan?

McCRACKEN: No, I was asking a question. The question is, should they now go out as salesmen for MAD? Alternatively, should they go to the ALGOL people and sell them on the improved features that they have developed in MAD?

PATRICK: Selling something to the Secret Society is a difficult job.

McCRACKEN: There is now a well established committee of IFIPS for the maintenance of ALGOL.

CLIPPINGER: I'd like to make a statement as the American representative on the programming languages committee of

IFIPS, who set up the ALGOL committee. The Programming Languages Committee met in Munich and set up an ALGOL committee. This committee includes the thirteen original authors who wanted to continue to participate. Some chose not to continue, for one reason or another. To this list was added representatives from many or most of the groups that had actively implemented ALGOL, so the group is much larger now. There are about thirty members and they have had their first meeting as an ALGOL group.

The purpose of this is to provide a much broader base to the ALGOL effort. It provides an authoritative body to answer questions and provide interpretations. It can extend the language and do whatever else is required to put ALGOL on a sound basis. It isn't meant to be a secret society. Anyone with a legitimate interest can probably find a way to get on this group. The original thirteen were polled at Rome to get their agreement on this approach. ACM on the one side and GAMM on the other side (they were the original sponsors of ALGOL) had wanted to get it into a broader group and moved it in this direction.

OPLER: I'd like to clarify some remarks that I think possibly Galler has misinterpreted. At the ACM meeting last year I spoke on what is happening to the effort on programming languages. At that time I estimated that we had a 3500-man-year backlog in automatic programming. This represents something like the entire membership of ACM working for six

months. The rate at which this backlog is increasing might force us to change the name of the society to The Society of Compiler Writers and Language Standardizers. At that time I stated (and perhaps, I was misinterpreted) that I thought that production of full-fledged processors by the universities would eventually taper off. By analogy, while the Electrical Engineering department does research on electronic components, they still leave it to the manufacturer to produce working components. I implied that the processors that we would use for routine work would be developed and maintained primarily by the manufacturers rather than by the universities. Heaven knows, we need the universities to develop the new ideas in the form of small breadboard models of new concepts, new languages, and new approaches. The manufacturers, however, should be allowed to pour the money into developing the big workhorse processors.

GRUENBERGER: One of the things that impresses me about the attempt to have a language become common is the very short half-life of such a venture. FORTRAN is a fine example. From time to time FORTRAN gets to be a little bit frozen. A new tape comes out from IBM; we all get a copy of the tape and we all have the same language for about 5 minutes. Then everybody goes off in different directions again. Our experience here at RAND is probably quite typical. We diverged from FORTRAN-17 (or whatever is currently kicking around) about a year ago. Isn't that right, George?

ARMERDING: Two years ago.

GRUENBERGER: O.K., two years ago. And we're miles removed from what everyone else uses. It's no longer common in any sense except that a certain amount of training can be transferred from person to person. But our FORTRAN codes can't run anywhere else.

GORDON: Excuse me Fred, but I'd like to ask you to refrain from using the word "everybody" to mean 7090 users. There is more to FORTRAN than 7090 use. However, if anything, that makes the situation worse than you just said.

GRUENBERGER: Of course, the same thing applies to FORTRAN among the 1604 users and everybody else. These people get a package and they're common with everyone else for a few milliseconds and then they think of a goody to put in or a patch to make and they're not common any more.

CLIPPINGER: I'd like to say a little bit about the process of standardization. We in X-3.4 who are trying to do something about it are not all convinced that we are going to succeed. Standardizing a programming language is an extremely complex business. Fred's point about half-life I think is quite pertinent. I don't think we are going to learn how to do it fast enough to achieve any results on any of the languages we've selected. This is, of course, just a personal opinion. And yet, we're hard at work trying to move in that direction. I feel a need for a von Neumann to get to the heart of the problem. We need a definition of the

syntax of a programming language. The right kind of brilliant man could lay a firm foundation there which might enable us to mechanize on a computer the determination of whether you have a properly specified language. He could provide us with tools to enable us to determine whether we're ready to consider extensions; to give us further information to enable us to resolve ambiguities. Without some sort of firm foundation I think we're just going to spin our wheels and the languages are going to move faster than we can move to catch up to them.

I'm sorry to have to be so gloomy about it. But I can't see standardization as a simple enough process at the moment with our present know-how to be able to keep up with it.

PATRICK: I think that is profound, Dick.

PHILLIPS: I'd like to comment on something that Dick Clippinger just said and take exception to it. He made essentially the same statement that we heard just now when he was a practicing member of the COBOL effort. He didn't think then that COBOL would ever get off the ground.

CLIPPINGER: Well it is off the ground but it's different for every implementation and we have no experience as yet in its use. I would guess that when the 1410 users discover what it is they have, they will be so sick that there will be a severe reaction. We'll get the same reactions from other small computer users (it has nothing to do with the 1410). The fact is that the language is so complex that it is extremely difficult to put it on any computer with a small memory.

GORDON: Some users of big machines are going to be sick too.

CLIPPINGER: There's at least some hope here. With a larger machine you have more degrees of freedom. You can improve what you first get (although it won't be good enough) and eventually converge toward something which is so good that the users will want to use it. But we have a lot of learning to do before we know where we stand or can evaluate what COBOL is really worth.

PHILLIPS: I'm not suggesting that COBOL has arrived, in any sense. I am suggesting, Dick, that you are surprised at our rate of progress compared to what you thought it would be two years ago.

CLIPPINGER: That's probably true. I think everyone here recognized the weight of that DOD mallet.

PATRICK: I'm not sure that's true and that may be why we're here today. I understand that the DOD says:

> "For an electronic digital computer to be installed on or after December 31, 1962, computers will be selected from those for which a COBOL compiler is available, compatible with the equipment delivery, unless it has been determined that the intended use of a particular computer would not benefit from the availability of a COBOL compiler."

I don't think you'll find a military officer in his right mind who would buck that. I don't think these boys have

guts enough to stand up to you and say, "Charlie, my compile time is gigantic and my throughput is way down. I can't stand it." I don't think they'll speak to you the way I will.

PHILLIPS: Remember I don't work for the government any more.

CUNNINGHAM: I don't agree on Bob's comment. I'd like to invite you to come and sit in my office some day and listen to the way our data processing people sound off about things they don't like. And this is certainly one of them. How, in talking about common languages, did we get away from that chart? We have really been talking about little segments of the chart at times when we talk about all four areas. The only portion in which the Department of Defense has a principal or semi-exclusive interest is the top one (Command and Control). We have a tendency to a variety of languages because there are different groups programming in Command and Control, leading to reluctance of one to use a concept developed by any of the others. But the problems in Command and Control and operational uses seem to me to be different from the general purpose problems. C and C languages are concerned with hardware systems built for a certain technological state. The costs and ramifications to the Defense Department in changing that system include the cost of changing the program also. When you talk about the Defense Department in relation to the scientific or business applications you have a different set of circum-

stances.

PATRICK: But the big push was toward the COBOL effort. I am concerned as an American in the Command and Control area and that's why Tom Cheatham, Jerry Koory, and Bill Dobrusky are here because they are the experts in that area.

GORDON: Did I understand Cunningham to say that the Defense Department is only interested in the Command and Control area?

CUNNINGHAM: No, no, what I said was that's the area in which we have a primary interest. This as opposed to the COBOL or Business systems (see your chart) in which DOD is just one of many, many users.

GORDON: An interest which you share with pretty near everyone else.

PATRICK: The secondary areas will take care of themselves because the directive is published and there isn't a man alive who is man enough to say "I made a mistake" and rescind it. I think this is the legacy we received from Charlie Phillips.

McCRACKEN: Are you saying that COBOL is a big mistake? There are also people who have gone the first round and are happy with it, so far.

PATRICK: I am saying that the requiring of COBOL <u>in its present state of development</u> could be a hell of a big mistake.

CUNNINGHAM: Well let me come back to that point. You read us an excerpt from a Defense directive. That directive

did not say that it required COBOL. It said that equipment to be furnished had to provide a compiler for COBOL.

PATRICK: True.

CUNNINGHAM: But that doesn't say that the installation is forced to use it. That doesn't direct anyone in the Defense Department to use it.

PATRICK: Do you think if they were bringing out the H-800 again they would do FACT in the light of that development?

CUNNINGHAM: I don't know.

ARMER: In the light of that development look what's happened to COMTRAN (Commercial Translator). COMTRAN and FACT are both dead.

PATRICK: Yes, in the light of that one statement. Now is that good?

CUNNINGHAM: From our standpoint in the Department of Defense yes, it is good. Not that either or both are dead but that we only have one language. It would be hard to estimate the cost to us of not having the ability to select updated equipment due to conversion costs. Common or standard languages fit the pattern developing throughout the Department of Defense in which the department will be dictating standard practices for use throughout the department. The tendency in D.P. now is toward commonality of procedures and variation of equipment. This orientation in the DOD will lead to writing procedures in something like COBOL rather than directing the use of a particular kind of equipment.

PATRICK: Don't you think that you're shoeing the

wrong end of the horse? It looks as though your problem occurs...

CUNNINGHAM: Our problem has occurred already.

PATRICK: The problem gets refreshed every day every time you order a computer. Thirty-six months from now you probably won't have anything installed that you have installed now. In other words you'll get a fresh start in 36 months.

CUNNINGHAM: Gee, I wish you'd talk to the Government Accounting Office (GAO). But, have you considered the conversion problem in an installation where there are between 500-600 routines (all dynamic) with over a million instructions and the problems and cost of changing these programs and equipment?

PATRICK: These are sort of facts to the computing field. How many UNIVAC-I's do you still have running and don't you wish you could get rid of them? UNIVAC-I's and 704's are about the only two machines that are still around that have been here more than 40 months.

CUNNINGHAM: There are still a lot of 705's, 650's, etc. that have been around a long time too.

PATRICK: Yes, but they're probably now mod 3's which are incompatible with the mod 2's and the mod 1's.

CUNNINGHAM: I don't know. You're talking about the whole Defense Department and I am not familiar with the situation. I can answer in general about some of the things you're talking about.

PATRICK: Well, I think your main motivation for a common language stems from the fact that you want to order one

computer from every manufacturer and set them side by side. This is bound to give you trouble. This is like trying to drive a stagecoach pulled by a zebra, a kangaroo and a mule, all in harness together and we're going to go to San Francisco. It doesn't seem to me that the solution of your basic problem is through a common language.

LITTLE: I think there's a point here though. You're expecting your common language to cut across machine types and classes. I know from experience in the Air Force that the different depots, for example, have different kinds of machines which are expected to implement the same inventory control system. What happens then is that you are not only forced to take the same problem and implement it for different depots, but you are then forced to make the language compatible across the machines too. This is a big job, the way things sit now.

GORDON: There are only three problems the way I see it. We're trying to standardize the wrong thing. We're doing it the wrong way. And we're doing it at the wrong time.

Let's take the way we're doing it. Traditionally, standards become recognized as such by virtue of the fact that they become standard through usage. After long use people look around and say, "Gee, this is a good thing; let's give it the label 'standard.'" What we're trying to do now is sit back and pull out of the blue sky something brand new, that we will call a standard, which will wipe out everything

that went before. This is why I say we're doing it the wrong way.

Secondly, we're trying to standardize the wrong thing, which is what Paul Armer said before. There are two different levels at which you can standardize. Bernie mentioned SAP before, which was a local standard, a standard among a small group of one-machine users. This was a standard; it just wasn't a world-wide standard. You had standardization within an installation, within a group of machine users, within a single manufacturer, etc. You had geographic levels of standardization.

Another kind of standardization lies in things like character sets, tape formats, and module standardization. Working your way up from there you can have standard functions, standard routines, and so on.

Perhaps what we should be doing is looking more to standards on the level of one-user, one-machine-type standardization. As Gruenberger pointed out before, even among the 7090 users (restricting themselves to FORTRAN) there isn't a single standard. And Paul Armer pointed out that even things as basic as character sets are not standard. Maybe we should concentrate more on this area and allow a certain amount of evolution to take place before we suddenly start at the far end--the way we talk to machines on a global scale for all problems. This latter might be the ultimate point at which to standardize. That's why I say we're doing it at the wrong time. I think

we might be as much as a decade early for tackling that problem. There is a lot of work to be done in the meantime.

BROMBERG: We <u>are</u> looking at character code standardization. You can't put on a set of blinders and close your eyes to all the progress that is current because it doesn't happen to start at the point that you have defined as the beginning.

GORDON: Unhampered by knowledge, we're going ahead and laying down some firm, rigid things that are likely to be a bit of a problem after a while.

BROMBERG: The way that we're approaching--the matter of commonness of a language--is perhaps untimely. I would maintain that there are two reasons for having a common language. The first is that it provides an effective means for the specification of problem solution. You can't really say anything against this. Everyone is looking for a good usable vehicle for expressing problem solution.

The second is this ability (that DOD is waving) of being able to take a given problem specification in this common language and run it on another machine.

Now standardization activity can still be done without considering the second objective; namely, the ability to take a program written for machine type A and run it on machine type B. That is primarily DOD's problem. It's a single problem that they have. This, in itself, should not preclude the activity that is going on among all the manufacturers. Our responsibility as manufacturers, (and it is a definite

responsibility to the users), is to provide this effective vehicle without regard to the ability to put a program written for one machine on another. That latter ability may come. It will probably come as a direct result of the standardization efforts that Clippinger was talking about.

GORDON: You talk about the ability to communicate effectively with the machine. You need a good problem-stating language. For all we know, FACT may be the best problem-stating language yet devised. But we are not going to have a chance to find out because it is not going to get sufficient usage. It will get some usage (as did Commercial Translator) but the blunt fact of the matter is that we are never going to have any really effective usage because of the fact that the standard preceded the evolution.

BROMBERG: Barry, in the creation of any one of these languages (the creation of the actual specifications themselves) there is never any of the wisdom of experience and use.

GORDON: That is correct. Consequently, some of them don't quite work out. I agree. You have to have a chance to find out whether they will work.

BROMBERG: That's what we are doing. Consequently, by finding out what their so-called standard does, you have every opportunity in the world to come up to the language maintenance committee and say "Look, this damn thing doesn't work. We have a function in Commercial Translator that is much better. Why not consider it?" What has ever come up

in the COBOL maintenance committee, for example? What sort of things have been suggested that were far superior in Commercial Translator? I know that there are some things. I know that there are such things in FACT also. Honeywell representatives screamed bloody murder to put a report writer in COBOL. Not because we had to have a report writer in COBOL, but FACT had it and COBOL didn't, and we had to keep up with the Joneses.

GORDON: A standard shouldn't try to keep up with the Joneses. It should represent a core, a nucleus.

You don't standardize everything to begin with. You standardize the core which can be agreed upon and then your standard grows.

OPLER: I think I have something for your "fact" bag. Sometime in the late summer of 1961, the SHARE COBOL committee solicited improvements in COBOL that people might want. Someone offered the following suggestion: Let's modify the COBOL statement that says "Add A and B and C to D" to "Add A and B and C to D and E;" that is, to permit the result of an addition to be put in more than one place. I think everyone agreed that this was a sensible suggestion. When the SHARE COBOL committee agreed on it and the SHARE Executive Committee approved it, the suggestion was forwarded to the appropriate COBOL committee. To the best of my knowledge, this suggestion (which was made in August of 1961) will not see the light of day in a COBOL compiler until late 1963 or 1964 if approved

by everybody down the line. Thus, a simple, logical suggestion takes about three years to appear in the processors that people use.

GORDON: Ascher, the feature you mentioned has already been in Commercial Translator for several years.

OPLER: Yes, I know that.

BROMBERG: To clarify, the proposal was submitted by SHARE in November of 1961. The COBOL Arithmetic subcommittee reworked it to remove many ambiguities in the ROUNDED and ON SIZE ERROR options. It was then passed by the full committee in March, 1962 and will appear in the forthcoming revision.

GALLER: There are opposing views being expressed here about the role of a standard. It seems to me that the stand taken by the ACM COBOL Maintenance Committee was: Let's not change it, let's give it a chance as it stands and see how it works. (I am paraphrasing their views, perhaps.) Now, with ALGOL, the language got implemented rather fast and hence things got frozen a little sooner. In COBOL the more translators that get developed, the harder it is going to be to make <u>any</u> kind of change. I'm not saying that making a change is either good or bad but just that it gets harder to make.

PATRICK: Bromberg said a moment ago that there are two reasons for standardizing. One is to have a language to communicate with and the other is that the DOD wanted to put the same problem on two machines. It seems that the first

could be handled by a standard language for communication purposes and the second could be handled by not putting two dissimilar machines back to back.

GORDON: The first requirement can be handled by any one of a number of languages or any one of a group of languages. If the impetus is to get a good programming language there is no requirement at all that it be the same one that is being used down the block.

BROMBERG: I disagree.

PATRICK: Why is that?

* : Which side are you on?

BROMBERG: Don't overlook all those costs that Gruenberger has on that chart. Suppose you grant that any one of these unknown languages is adequate for the job. You have training costs to begin with. If you're using two different languages you have to do twice as much training as though you use one language.

PATRICK: Only when I interchange people.

McCRACKEN: May I get in here? I heard several times the statement made that only the DOD has the problem; that they are the only ones that have different machines that have to talk to each other. This is not true and I'd like to offer a small fact. I know of a certain user who had a machine, let's call it Machine A, (Model 3, as a matter of fact). One machine, one user. They needed a bigger machine, so they surveyed the market among the various machines available.

One of the available machines was Machine B, made by the same manufacturer. Machine B would accept the programs of Machine A. Their evaluation of the machines involved indicated that they did not want Machine B. It was not the best machine, for the price, for their job. They ordered Machine B anyway because they figured they couldn't afford the reprogramming cost, which, by their estimate, was about half a million dollars. So they went ahead and got the machine that they didn't want just because of this non-common language business. It isn't just DOD.

PATRICK: I don't see where that has any bearing at all.

McCRACKEN: If they had been in COBOL in the first place they wouldn't have had this problem.

PATRICK: If they had been in COBOL in the first place--present state-of-the-art, now--they might have been spending twice as much to get their programs in.

* : And if they had been in COBOL in the first place they might very well have been getting the language they didn't want just because it was standard.

GORDON: And also they might have needed the added capacity of Machine B sooner.

GRUENBERGER: I think all that Dan was pointing out was the rebuttal of the statement that only DOD has the problem. We have seen examples lately of other people having the problem; Westinghouse, for example. They have recently stated that they are going to use COBOL company-wide.

CUNNINGHAM: In the first place the COBOL effort is made up of a lot of different people besides Defense, so a lot of other people must at least think they have the same problem. I'm not speaking only of manufacturers. There are a variety of users in the COBOL effort who must recognize a need and be interested in getting on top of it.

Secondly, I don't think that you'll find, even in the Defense Department, two different machines in the same installation back to back doing the same job. You won't find the 501 and the 705 working back to back. What you will find is the 501 in one place and a 705 in another place.

PATRICK: What is at David Taylor?

CUNNINGHAM: I don't know.

* : David Taylor has one of everything.

PATRICK: It seems to me David Taylor has a LARC, a UNIVAC, ...

CUNNINGHAM: I presume that the variety of work dictates a variety of equipment. I'm saying they're not doing the same jobs back to back.

ARMER: How about logical back to back, where you have one machine at one site doing, say, inventory control and a different machine at a different site doing the same inventory control problem?

LITTLE: You can still be hurt by having different

machines in the same installation because you do not have the freedom to pick up the load of one from the other.

CUNNINGHAM: Agreed we've just gone through the same analysis in the Air Force that Dan was talking about; the one that says that we didn't want to pick up a new machine because of the programming investment involved. A machine that we might really want we would never get to because we couldn't afford the reprogramming costs (both dollar and time). The question we face is, "How much longer can we afford to pay what might be four times the cost of the job to wait to get to the optimum position?" So we decided that the optimum for the moment is this swap we went through when we changed 705's for 7080's.

LITTLE: There's a very good point here. Is a language really going to solve this problem? Do we really design languages for use by what we might call professional programmers or are we designing them for use by some subhuman species in order to get around training and having good programmers? Is a language ever going to be an effective substitute for really good people?

* : Is there a difference between those two groups?

McCRACKEN: It won't be perfect but it will be better than what we have now.

LITTLE: This isn't clear to me. On small jobs you can stand inefficiency of all kinds. It's not at all clear to me that on large jobs like Command and Control that you can gloss over inefficiencies. I'm not sure that my own corporation--RAND--or the universities ever face this kind of a problem. Therefore, I'm not sure that the research that is going on in places like RAND and the universities is actually facing up to the problem. In these really big jobs hand coding will push the limit of the machinery that we have available today.

CLIPPINGER: I'd like to talk to McCracken's point. One of the reasons that users would like to get a good programming language is the freedom to change from one machine to the other. Dan pointed out that the switch is pretty tough when you are in assembly language. Now there's a fact here that I think most of us would agree to which seems pretty obvious but it ought to be stated. That if problems were formulated in an English type language (COBOL, or FACT, or Commercial Translator), that the statement of the problem contains all the information necessary for the computer and therefore contains most of its own documentation.

McCRACKEN: A very good point.

CLIPPINGER: Now if you want to switch that same problem from COBOL to FACT, or FACT to Commercial Translator, or even

from COBOL A to COBOL B you have about the same order of magnitude of work which is ten times less than changing from 705 Autocoder to some other assembly language. So there seems to be an intrinsic value in the English-type language. This advantage could even be gained in a language that is not English narrative as long as it's some symbolic language that is suitable for stating problems. The point is that compilation produces documentation. We're all aware that the big problem in moving a computer solution from one machine to another when you are dealing in assembly language (we're talking about very large problems) is that the people involved in producing that checked-out code are no longer around; they may have changed jobs or maybe gone to different jobs; and that essentially you don't know what those checked-out instructions really do. The programs have probably been patched and repatched and when you come to rewrite you can find no one who really understands what those instructions do. It's very difficult to get that kind of information when the problem is stated in assembly language and it's frequently easier to scrap the whole thing and start all over.

GALLER: You may have the same problem with English-type languages too.

PATRICK: Yes, it's not clear that you've changed it any. If you have poor management and you haven't kept your

source decks up to date you have the same problem.

CLIPPINGER: That's debatable. It depends on how you do it. The problem can exist.

PATRICK: With long compile times the problem statement will exist in two forms.

CLIPPINGER: With FACT, for example, you compile and you don't patch because it's too difficult to patch. You simply recompile when you redo the problem. Therefore the final version which is running is supported by a document. And you know exactly what that program does. It's my guess that COMTRAN handles this the same way. In the case of COBOL there may be variations from installation to installation but I'm pretty sure that in most cases COBOL is treated the same way.

PATRICK: Howard, is it true that in 501 COBOL you patch because the compile time is long?

BROMBERG: I guess one can do whatever he pleases in 501 COBOL, as far as the object code is concerned.

There's an interesting thing about language design itself. In the COBOL area, for example, there is nothing in the language itself that talks about, or makes provision for, documentation. The language designers apparently believe that the language itself is sufficient for documentation. There is nothing built into the language apparently to accommodate this notion of debugging, or fast recompilation time. It's just not there. It is a shortcoming.

Little raised the question, "For whom is this language designed?" It is my opinion that a language like COBOL is designed for two kinds of people. First, it is designed for the implementers--the guys who are actually going to interpret the language specifications for a particular computing device. Second, it is designed for the salesman, so that he has something to go out and talk about. It is not really designed for the user, per se. The secret for the effective utilization of all these languages is the recognition by the user that he cannot exist in the common programming language cosmology as a clod.

GORDON: I'd like to tell one small anecdote with reference to the guy COBOL is designed for. At a COBOL session we had not too long ago we were discussing the wordiness of the language. COBOL uses words like ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE. With the COMPUTE verb, they allow PLUS and MINUS for the ampersand and hyphen (which were considered too mathematical at the time). Thus, you were allowed to write "A MINUS B" (all spelled out) and someone suggested that the committee should also allow the term "TAKE AWAY."

McCRACKEN: I have a fact. I am strongly impressed by the remarks that Bromberg made that COBOL can't be used by clods. I ran a little program to prove it. It was a little COBOL exercise designed to prove just that. I ran it on the 1105 at the Air Force Logistics Command. There were two

versions, both of which had exactly the same procedure division. They produced exactly the same results. The only difference between them was in the data division, and the change was minor.

We compiled the first one and the running time was about 8 seconds. The second one, which was also a legitimate COBOL program (a slightly modified data division, the same procedure division, and the same results) took 80 seconds to run. I think this goes to show that you can't be a clod. You do have to know something about machines. It also goes to show, I think, that the efficiency of the object program is not just a function of the language, or of the compiler.

I think it also shows that you don't have to know very much about the machine. I had never worked with an 1105 before. All I used in designing this horrible example was knowledge of how alphabetic information is stored in a binary machine.

BROMBERG: I would bet that a competent 1105 programmer could make some savings again on your first example.

McCRACKEN: That's quite possible.

GORDON: I'd like to thank McCracken for his fact, and point out that the point that he made was not the one he claimed to make. It seems to me that he has demonstrated that COBOL can be used by clods, but with disastrous results. But this brings up another interesting point. Isn't it the responsibility of the manufacturers and the language designers to come up with

a language which will encourage effective use of equipment rather than simply to enhance the sales and allow anyone to put a program on the machine?

LITTLE: Doesn't the language then gloss over differences in the machine? Doesn't it sort of hide the machine? In some sense the statement you just made is speaking against languages.

GALLER: Al Perlis once ran an experiment (I can't remember the details of it exactly); what it amounted to was cutting a program down from 20 minutes to 1 minute on some machine simply by moving things in and out of DO loops and computing a couple of things in advance instead of within the DO loops. It has nothing to do with the machine and nothing to do with the language. It is simply an appreciation of the structure of a problem and the use of good organization. You can get people to do this no matter what tools you give them and you can get people who will do it wrong no matter what tools you give them. It's a problem in education.

GORDON: There are situations, Bernie, that do have nothing to do with the language, or with the machine. I say that there are also situations in which the more "sophisticated" languages get, the more the language can enhance this sort of thing. I have seen 7090's which pay a very large penalty in trying to simulate 7080's. Some clod has simply taken a program written in a common language for one machine and run it on another and he's gotten results. I have seen 7090's operating at like 701 speed in order to do this. It's

probably true that there are programming considerations independent of language, but it is also true that some of the newer languages, by masking the machine more thoroughly, have tended to encourage greater inefficiency.

MacKENZIE: I agree with Galler that the problem is one of educating the users, and I suggest that it should be so constrained. Consider that good programmers using a so-called machine-independent language will tend to construct machine-dependent algorithms, even though they don't think of them that way. This point might be overlooked in the argument that McCracken advanced--that there is this latitude, if I might call it that, in the language which the programmer might exercise to the advantage or the disadvantage of his installation. If there were not this latitude, we would probably find, what were in fact, highly machine-dependent languages being passed off as machine-independent languages. I think it is important that there be this latitude. The real important question should be,"How powerful is the language?"--and one should presume intelligent use, arrived at possibly as the result of proper training.

LITTLE: I agree that it's important to recognize that this exists. I'm not sure but what these languages aren't designed to be used by lower level people (in two senses). First, there is the guy who is at a lower level intrinsically (who would never become a top notch programmer) and, on the other hand, there is the fellow who might become a good

programmer but you want to get him doing useful work sooner. If you don't recognize that this latitude exists you may be letting yourself open for tremendous problems in your running time and your problem organization and this type of thing.

MacKENZIE: I'm not trying to be argumentative, but don't you think the same problem exists at the machine language level? I can think of all kinds of examples where there was a factor of 10 difference in an object program's running time. For example, undoubtedly Dan could have achieved that difference in the procedure division alone.

GORDON: The complexities of the language you use may add to this and if you add to it sufficiently your weekly payroll may become a critical real-time problem.

PATRICK: I thought Bromberg had covered this pretty well in his Datamation article. In that document he upset once and for all the idea that these dumb languages are going to cut down your training problem. He maintained that you had to know the language and its implementation and the computer in order to use it well.

McCRACKEN: Now wait a minute. You don't have to know as much about the computer as if you were going to do it in, say, Autocoder.

PATRICK: O.K. I don't have to know as much about it but I can't do it as though I were a complete idiot.

GORDON: Who are "you" when you're talking?

PATRICK: I'm the guy who is writing instructions.

BROMBERG: There are many guys who are writing. Let's consider an installation of 15 men. Before the advent of these English-like languages the 15 men each had the requirement to have detailed knowledge of the equipment. What we're saying now is that, being removed from the details of the machine, this requirement is less stringent. Effective installation practices can now allow one individual--out of these 15--to be the machine expert. That one expert can act as the consulting programmer.

PATRICK: But wasn't that always true? We never did have uniformity in our shop. There was always a priest who really knew all the equipment and the other guys kind of plodded along.

LITTLE: If you have one good guy and 14 others who just follow along, who takes the place of the good guy when he moves up?

PATRICK: The trouble is with these languages the good guy has to be about three times better. He's got to be really top notch.

GORDON: He's got to know the machine...

PATRICK: And the compiler, and the language.

BROMBERG: But especially he's got to know the machine. The secret in the area of COBOL, for example, in the generation of efficient object programs is in a proper utilization of the data division. This means that you must know how your computing device handles and manipulates data. It becomes

the function of this one knowledgeable man we talk about to be the data describer for the installation. You recognize, of course, that in business data processing you are going to manipulate only a small finite number of files. His job then should be to set up initially, with the computer in mind, data divisions for each one of the files, keeping in mind the idiosyncracies of the compiler. He then becomes the file clerk. We call him a librarian. Then every other programmer writing a program which uses these files consults this librarian and is guaranteed, to a definite extent, that he has the most effective efficient description for his file.

PATRICK: This is what SDC does with their COMPOCL.

BROMBERG: This really has nothing to do with the language specification. This is just good efficient installation practice. We can't interject this into a language design. As Gordon suggested before, a language should enhance some of the computer's features. That's nonsense. Why should it? The function of a language builder is only to assure himself and his corporation that none of the features that his equipment has will be precluded by a language specification. There are really two languages. One is the language that exists in the infinite world of specification. And the other is the interpretation of this language which is the finite world of the processor. Clearly, (for example, in the COBOL specifications) it says that you can have a literal of any size but you can't deal with a literal of any

size in a specific processor. We will allow infinite length literals if someone will tell us how to write them.

OPLER: I was talking to an installation manager who uses COBOL. He decided to adopt COBOL because he figured that his problems divided into two classes: those that would fit into COBOL and those that were rather unconventional. For the latter, his best programmers would write in assembly language. They went ahead on this basis, and found to their consternation that the following situation had developed: the programmers who were trained in COBOL (but who did not know much about the machine), were writing programs in COBOL. After compilation, the object programs would not run. They would then call the other group of programmers to patch the COBOL object program. Eventually, they found that they were using both teams full time on COBOL. Maybe this will be the final division of people in such installations. Groups A and B will sit on top of each other to help each other.

CUNNINGHAM: I have an analogy to what you just said, Ascher, but I am reminded of our fire in the Pentagon. I had a call from a user who sounds very much like the man you just described. He offered me 20 hours a day on a large scale computer. I thanked him profusely and hung up and said to myself, "Using the computer only 4 hours per day--it's obvious he doesn't work for the government." I think there's the same degree of management efficiency involved in each case.

PATRICK: There was a lot of spare machine time that we didn't know about. It came out when you had that fire.

BROMBERG: On the other hand, there are many cases where people are using COBOL and using it very well and they have never compiled a COBOL program.

GORDON: That's probably the best way.

OPLER: That sounds like the story I heard of the manufacturer who "ordered" a big computer to take over all the plant functions and, two days before it was scheduled for delivery, he cancelled it. The truth was that he had never planned to install it in the first place.

LITTLE: Somebody said a while ago that such arguments shouldn't be interjected into a debate on common languages. I don't really think it's nonsensical. I don't think enough of this sort of thing is interjected when people actually sit down and do the design of these languages. I think either a selling job has gone way too far or people don't realize the degree of latitude that is in there. If the latitude exists then they should be educated to it.

BROMBERG: I'm just griping about the use of the word standard. I don't really know what it means in this context.

GALLER: I'd like to ask a question. Suppose that the DOD goes ahead with this idea of standardizing our language. Can we predict what will happen to ALGOL, if anything? Of course I include in that question JOVIAL, FORTRAN, NELIAC, MAD, and all the rest of them. We see COBOL and we ask, "What

happened to Commercial Translator?" They more or less arrived at the same time and maybe that makes a difference. FORTRAN is pretty well established. ALGOL is pretty well established as far as it's gone. What will be the effect on these languages? Should there be an effect?

CLIPPINGER: Are you asking whether standardization has an effect on these languages?

GALLER: Well, will DOD's choice of a language be, in effect, standardization?

McCRACKEN: Would someone please fill me in. Will DOD's pending decision involve a language like FORTRAN?

CUNNINGHAM: What decision are we talking about?

PATRICK: A Command and Control language.

CUNNINGHAM: I don't know.

GALLER: One of the things I read in preparation for this meeting mentioned a language like JOVIAL that might evolve.

LITTLE: I think the question is (to further the discussion here) "What would be the effect if the DOD adopted JOVIAL tomorrow for the Command Control language?"

KOORY: I'd be very surprised.

I have read the document that was distributed (SDC Manual TM-688) in many forms. There were at least three drafts before the final version. I've liked it better in each version, perhaps because I'm getting used to the idea. If you want to talk about standards I'd like to go back to

the point that Gordon made much earlier in quoting from Dick Hill. I think it's very important that standards evolve; that they are not set. I think that you might want to establish a study activity of some sort to look at the problems for a particular area, and to decide how best to describe these problems. You can perhaps define a language in which we can describe the problems of Command and Control. I would not suggest that we take anything that exists today (within my knowledge) and say that that ought to be a standard. I think that would be a mistake.

ARMERDING: As the standards evolve have we any guarantee that we're not going to come up with the English system again, rather than the metric? Unless we have top notch people working on it we're likely to come up with the English system all over whether we like it or not; and then it will be too late to change.

PATRICK: That may be true but we're going to have to broaden the language design boys a little bit. They can't just think of producing a language that is easy to describe without keeping in mind the fact that they are building a tool. All the things that we have listed on this round chart will be affected, and will affect the design. If you pick a language that is impossible to train up to (that is, one that the lower level people could never be trained to handle) then that language is not going to go.

McCRACKEN: What do you have in mind?

PATRICK: Let's think of the Civil Service types who have to be trained up to use all the data description devices in JOVIAL (partial word field manipulations). You may be able to do something to mitigate this problem by writing some good training literature (which is somewhat lacking today). But it's really asking too much.

GRUENBERGER: Look at the world we just opened up for you, Dan. There's another book to be written.

LITTLE: I'd like to get back to the subject of evolution of standards. I go along with most of what was said but I think that we're overlooking one fact--that there are Command and Control jobs that have to be done. If, in the process of implementing these jobs that have to be done, the Defense Department (or any other large user) decides to standardize on a language like JOVIAL, then, like it or not, we're going to be sitting around talking about JOVIAL the way we have always talked about FORTRAN.

PATRICK: And in that situation a subset of JOVIAL would do the FORTRAN job very nicely.

McCRACKEN: Would it?

PATRICK: I think so.

GORDON: You'd better put that one in the opinion bag.

PATRICK: All right, it goes in the opinion bag. You have to assume some sort of utopian society. You might have to assume that there is no investment in FORTRAN, which we know is wrong. There might be a hundred million dollars

invested in FORTRAN both in people training and in translators. This may be too utopian to be possible but if all other things are equal I think a subset of JOVIAL or a subset of NELIAC could do the job. Both of these languages are more powerful than FORTRAN and I think both of them could do the simple, straightforward, scientific evaluation jobs well.

Of course, this isn't the world we live in where we can throw away a hundred million dollar investment.

GRUENBERGER: George, are you going to sit there and accept that remark?

ARMERDING: No one yet has said anything about compile times.

PATRICK: We're talking only about the language at the moment.

ARMERDING: And I think that's one of the main troubles with people who design languages. They seem to forget about things like the compile times and execution times.

* : Not intentionally.

ARMERDING: And when they finally see the result running on their machines they throw up their hands in horror. As Dick Clippinger said, when the 1410 users see what COBOL is going to do for them and to them they're going to be horrified. You can't really predict this. You can't say what the effect of COBOL is going to be on a machine that hasn't even been announced yet.

LITTLE: I think I'd feel a lot better if I believed that

some of the users would have enough sense to scream. Some of them are going to put a high level language on a machine and not know any better.

ARMERDING: Yes, that's also true.

PATRICK: Barry, do you hear any screams?

TALMADGE: You'll hear some screams from users of FORTRAN on the 7070, but there are an awful lot of 7070 users of FORTRAN who don't scream; and this worries me more. Quite a bit more.

* : It's quite possible that they're getting something worthwhile out of it.

GRUENBERGER: There are 705 users who are mighty unhappy over FORTRAN.

DOBRUSKY: We're talking about compile time on these large compilers. Agreed that they're much longer in their compile times than other processors--I'm referring to COBOL, FACT, JOVIAL, and so forth. At SDC studies are being conducted among JOVIAL users (not only within SDC but at other places as well). None of them have complained about the compile time, perhaps because they have too many other things to complain about. That was a fact. This is an opinion. One of the reasons they don't complain about compile time is that they realize that there is nothing else available at their hands, right now, that can do the job as well. Their experience with FACT, NELIAC, FORTRAN, and what have you, indicates that JOVIAL is better than most.

* : We were talking about both compile times and run times. They do complain about running times, is that correct?

DOBRUSKY: Yes, but what is program efficiency? We talked about the old trade-off between space and time. What is a good program? Is it the amount of space it uses or the amount of time it takes? With a POL we address ourselves to that problem and to a much more important one; namely, the elapsed time involved from the formulation of the problem to the finished result. If it's a real-time problem then I say there's not a compiler in existence today that will produce you efficient code for this application. As Dan pointed out, using the power of a language by the unindoctrinated can produce pretty horrible code. It's just like many people are able in common English prose to describe very crisply in one sentence what it might take someone else a whole book to describe.

PATRICK: Be careful what you say about books, with McCracken around.

GORDON: I take exception to your statement, George, that language designers do not consider compile times. Some of them do; that's a fact.

ARMERDING: They can't if the machine hasn't even been designed yet.

GORDON: Sometimes you're lucky if you're working on a language for an old machine.

GRUENBERGER: You mean like one of these days we're going to get FORTRAN for the 7090, huh?

GORDON: Can we delete Fred's remark?

Basically language designers will take into account (as best they can) things like compile times. But there are many other things they have to take into account. For example, when you design a language, some poor slob is going to be stuck with the problem of teaching it to people. There are problems of implementation schedules to be met. You aren't going to design something that you can't implement for the next thirty years; not deliberately, anyway.

So there are many things that have to be taken into account and I think that compile times is one of them. Probably Dick Talmadge could tell us something about compile times. I know that Commercial Translator was designed with compile time in mind. It turns out that for two of the three machines for which Commerical Translator is running, compilation was complicated by other considerations; i.e., other than the language itself. For one totally new effort with Commerical Translator (namely, on the 7090), I think the compile times are quite respectable, to put it mildly. This is something that is taken into account but unfortunately it can't be the only thing. There must be compromises.

ARMERDING: I agree. For example, compile time is not nearly as important as efficiency of the object code. You are probably willing to buy a certain amount of long

compilation if you'll get out an efficient code at the other end.

ARMER: Sometimes you have one goal in that equation of Patrick's, sometimes you have another.

ARMERDING: Each user is going to have to put in his own coefficients but some of them are locked in whether he likes them or not. I can't change the FORTRAN compiler on our machine so that in some instances it will compile faster at the cost of the object code and sometimes the other way around. I just can't do it. Those things are locked into the compiler.

ARMER: In the first version of FORTRAN they devoted so much attention to object code efficiency that the compile times were way high. That was on the 704. Now they seem to be going the other way, on the 7090.

TALMADGE: I don't think it's true that you need accept long compile times in order to get good object code efficiency. Very often, both in the scientific and commercial field, the life of a job is only one or two runs. In this situation, it is most important to be able to connect small parts of a program rapidly; that is, compile efficiency is much more important than run efficiency. This point was kept in mind in designing Commercial Translator.

As to the comment that a language can force a long compile time, I believe it to be true to some extent, but not to the extent many people believe. Too often the processor designer

loses sight of the fact that his system is going to be run on a particular machine, in a particular environment. It is more important to design an efficient, simple, total operating system than to turn out a super efficient object code rather than just an average object code. Most of the lost time comes from changing tapes and from the operators helplessly wondering what went wrong with that particular program. You may lose 20 seconds during an object program if the code is inefficient, but you could lose 5 minutes or 10 minutes during the entire job if you don't have a good operating system.

PATRICK: If I make a compiler efficient in the way you just mentioned I make it machine-dependent again.

TALMADGE: You're talking about a processor now.

PATRICK: If I take an easy language and restrict all my symbols to six characters or less so that they'll fit your machine then I can't run Clippinger's programs which were designed for his 48-bit machine. And it makes your compiler a great deal faster: that is machine-dependence.

BROMBERG: On the other hand, there are some of us who believe that one of the functions of a compiler is to do error-checking. We can certainly design a fast compiler (call it a User Beware Compiler) that lets you effectively write anything that you'd like. And you can thereby increase compile times by a factor of 10 and still keep it within the area of machine independence.

TALMADGE: If you're talking about computer independence

then you're talking about the language description. But you can't talk about computer independence when you talk about the processor which translates from that particular language to produce an object code on the machine. Then one cannot be independent, any more than one can be machine-independent when writing a data description for a commercial problem. Dan's example is a good one: When one writes the data description to take advantage of the binary machine then the object program is much more efficient.

I can speak personally for Commercial Translator. In writing it we used as many machine-dependent techniques as we could, in order to get the most out of the 7090 for the CT language. It is up to each implementer to do the same for his particular machine, since some techniques are good on one machine and produce a good compiler, while others might produce a very slow compiler.

PATRICK: The point I was getting at was that if you are implementing a processor you must be doing it for a specific machine. There is a threshold point at which it makes a great deal of difference whether you put restrictions on the language or not. It's the same language; the syntax is the same, the verbs are the same, it has precisely the same meaning, but if I can put some restrictions on the use of that language by the source programmer it will make a lot of difference in the performance of the compiler.

TALMADGE: I agree with you on that, Bob. There are

certain things which are critical as far as compilation times are concerned and which buy practically nothing in language facility. The point has been made many times that one shouldn't try to be too general in a given language because the cost of generality is frequently high in terms of compile time.

The point I'd like to make, however, is that I don't think this is true until one gets to a fairly advanced stage in the language development.

PATRICK: But they're not machine-independent anymore.

TALMADGE: In what sense?

PATRICK: Well, take the example I used before. If someone has used 30-character names from the COBOL made for Clippinger's machine, I can't compile them even though they are supposedly written in the same COBOL.

TALMADGE: Well, we have 30-character names in Commercial Translator. I don't think it will have made any particular difference in processor efficiency if we had had 6-character names or even 2-character names. Again, this is a matter of technique. Granted some things, such as infinite length literals (mentioned a while ago), would be very difficult to implement. But that's because they can't be described well or are not reasonable to do.

DOBRUSKY: It seems to me that this discussion right here borders on the difference between standard and common. The fact that they're standard means that both of you can

handle them in one way and another. If you write them generally machine-independent, (acceptable within the constraints of the grammar and syntax of the language) you re going to pay for it on any machine. This is typical of every compiler and every language. However, I am sure that there is a standard subset of COBOL that both of you could use; the same is true of Commercial Translator.

It gets down to a matter of stock items again. An example I've often used is that of wire sizes. Wire sizes are predicated on certain attributes. These are commonly accepted and stocked; you can buy them anywhere. If I want some #17 1/2 wire, everyone knows what I mean; it's somewhere between #17 and #18. If I want to use it because of my particular application, it's going to cost me. It seems to me that continuing this idea of machine-dependence and machine-independence, there must be to some degree commonness by usage, whether it be in power or as a subset of the accepted language.

OPLER: I think there are several people in this room who have had the experience of designing two or more processors for the same language. I believe I can outline an argument which tends to disprove machine-independence.

Consider the definition of the language. In attempting to write the processor, the design group comes to a question like this, "What happens if there is an overpunch in the fourth character of the message?" etc., etc. They have to come to some decision. Now, I maintain that for a reasonably

granting the fact that we're talking about the same language?

BROMBERG: There's an easier answer and that's the fact that there is no authoritative maintenance body; that is, one that is both knowledgeable and peremptory.

CHEATHAM: Ascher refers to the way that ALGOL and JOVIAL are described, which is really quite formal. I don't think that's a really significant part of the problem. The decoding of the intent of the message is not the difficult part of getting an efficient code out.

OPLER: Would you like to explain what it means in COBOL when you meet the message "ADD ALL 43434343" and you name some particular variable which has a complicated data description? Would you explain how your COBOL processor is going to treat this case? That's an example of what I mean.

PATRICK: The COBOL language is not completely specified to the level you referred to.

OPLER: It's just like the map of Antarctica where there is a place that indicates terra incognito. How are you going to make a decision under those conditions?

PATRICK: Tom, do you have complete control for these same decisions for the three cases you mentioned?

CHEATHAM: No, not complete control. There are two things involved here. First there is the question of interpreting what are the formal specifications of a language. Formal specifications which are, for example, usually published really aren't specifications of the language at all. They are specifications of the class of languages of which

you are choosing one, by making these n decisions that Ascher speaks of. Then there is the problem of getting efficient compile times and efficient object code. These are quite separate problems. I was addressing myself to the second of them.

PATRICK: I'm sorry. We thought you were addressing the first.

MacKENZIE: A few minutes ago Galler brought up a subject which I think is a very interesting one. He raised the question, "What price do you pay for standardization?" I think this is interesting because it gets you back into areas like the trade-off mentioned on that chart.

It's my opinion that one very obvious price you pay is that of inhibiting the future, in some respect. Whether this is good or bad, you don't know. This is one of the principal problems involved in dealing with established languages on established machines. No matter how hard the language designers have tried they haven't been able to imagine the way the world might be at some future point in time. You do pay an apparent price, I'm sure. Yet, there are some very obvious, good things to be gained by standardization, particularly at the documentation level.

GALLER: I'd like to recount three anecdotes. One concerns the meeting in Paris in 1960 at which ALGOL was born. The people involved pretty much decided that not every function should be recursive; that they should be declared by exception. Peter Naur wrote up the report and

forgot to put this in. I think you all know that now every recursive function is pretty much a sacred cow that no one is going to touch. Everything that is done now has to work with the fact that every function is recursive. That's one anecdote.

I was at a SHARE FORTRAN Committee Meeting where they were arguing over changes that they'd like to see made in the new FORTRAN. There was a tremendous reluctance on the part of IBM to accept some of the changes that were being suggested by the committee. They were questions of compatibility with earlier versions of FORTRAN and FORTRAN for other machines and so forth. Bill Heising was there to protect compatibility. Various such matters were referred to Heising and his attitude was, "We'll have to look at it to protect compatibility." The claim was that compatibility had to be protected because there was so much investment. There was much discussion and many hard looks at the ability to convert programs from the old FORTRAN to the new. It was agreed that one could write converters to change from the old to the new and that this was all that had to be protected. That was really the only cost involved in order to go in new directions. We finally convinced the SHARE Committee itself to go ahead with the work, and the converter from the old FORTRAN to the new is pretty well checked out by now. As a result of this work IBM has made changes in the language of FORTRAN-IV which make it quite incompatible with previous FORTRAN's.

Anecdote #3. When we put MAD together, we had quite a few objectives but one of them concerned itself with the objection that we anticipated; namely, what are we going to do with FORTRAN, ALGOL, and so forth? Why should we switch to MAD? It's going to cost us a lot. Other people asked, "Is FORTRAN going to be a subset of MAD?" We took the position that, as far as possible, FORTRAN and ALGOL would both be isomorphic to a subset of MAD. In figuring out how to specify each statement in MAD we asked ourselves how could we translate ALGOL into this statement? We didn't have to compromise too much to provide for this translation. As a result we had a language which contained the ability in theory to translate from ALGOL to MAD. We also have on our master tape the ability to translate from FORTRAN to MAD and we use it. My point is, we worried about this $100,000,000 investment also. You don't necessarily have to protect your investment to the extent that programs will run without change on the next machine. It may have to run with change, but if it's a one-time change it doesn't necessarily have to cost too much. Preferably the change can be made on the computer. To be able to do this means that you have to anticipate the need, when you're designing the language, to be able to make this translation.

So I don't really feel too bad about seeing another language come out (even if it gets to be fairly standard) provided that I can see a way from getting from the old

to the new. When we rewrite MAD, if it's different enough, we'll call it something else, but we will provide as one of our main objectives a translator to allow people who are using the old version to work with the new. We don't anticipate that it will cost us very much to do this.

MacKENZIE: Talmadge made a statement a little while ago that I thought I understood at the time; namely, that you pay a price for generality. This is really due to two things. One constraint exists at the hardware level and another exists at the programming technology level.

The point I'd like to make is this: that undue standardization tends to inhibit development of either machine organization or, for that matter, technological improvement of programming techniques. I think, therefore we should consider very carefully what attitude we should have toward this thing. The anecdote that Bernie told about recursive procedures is a very good example. It's very easy to understand why people do not want generality and do not want to allow recursion in all procedures. They object primarily because of constraints that exist at the hardware organization level. Is that really the way to look at the question? Why not look at the question of "Why should we not have recursion?" and thus identify perhaps why it is not practical to have it now. My personal point of view is that we do ourselves a sad disservice if we don't take this latter point of view. I use recursion simply as an example.

GALLER: I'd like to point out that the B-5000 gets around the recursive problem very nicely, and I think it's wonderful. The reason I mentioned recursion as a thing that we probably don't want is that it costs so much.

MacKENZIE: You do pay a very great price for it in a conventional machine organization, but that wasn't my point in bringing it up. My point was, "How ought you to look at these things?" Should you be against them because you can't do them with the present technology or should you attack the present technology because you can't do them (and they are really worthwhile)? I didn't mean to imply that you ought to treat all procedures recursively, but you ought to be able to treat any procedure in a completely general way and if you can do this, you will find that most of the recursive problem has disappeared. The fact that you can't in most languages is indicative of the state of our technology.

GALLER: Unfortunately, there is a time lag, though. It's one thing to say that we have to change the technology to meet it and another thing to realize that we have to run these problems now.

CLIPPINGER: MacKenzie made a point which, incidentally, turned out to be a minor case against standardization. He wanted to have the freedom to attack the crucial points. I agree with what he said but I arrive at somewhat opposite conclusions. But I am motivated by the same desires.

Some of you may not be aware that the FACT compiler has 230,000 instructions in the processor. I asked myself, why

does it have so many instructions? Need it be so complex? That's easy to answer. "No, it need not be." Much of what is there is there because of lack of standardization. For example, 30,000 of those instructions are concerned with the card editing generator. Why is it 30,000 instructions? Partially because we allow very elaborate editing of the input information, because we regard it as extremely important to be able to purge the data that is coming into a data processing system. But partly there are 30,000 instructions because of the many different ways people use a card to store information in practically any way that people could have found it expedient to store information on a card. A little more discipline regarding the way you put information on cards (some agreement, some standardization on what you will allow) would have made it possible to make that particular portion of FACT considerably simpler. I'm sure there are hundreds of places where a little discipline (choosing one way instead of allowing all possible ways) would have made it possible to accomplish approximately the same results with a compiler that is much less complex. So I think there is a good case for standardization although I don't see clearly what it is. But I feel intuitively that it is there.

MacKENZIE: Quite obviously, I didn't want to make a case against standardization. I was simply attacking the basis on which a case for standardization must be made.

For example, in a lot of programming systems today it's hard to say that you haven't paid a price in many respects,

for not allowing generality. To take a simple example, how many processors today restrict indexing to just three levels? How much in present day processors is there simply because the designers were checking for adherence to their restrictive notions? Some of the programming systems that we have done have been rather compact at the processor level, due in part to the relatively high degree of generality present in our languages. That may be a surprising thing to say but I really feel that it is true. Sometimes, in the interests of arriving immediately at standardization, we all descend to arguing about how our present machines can handle the proposed language or how well aware we are of how we might proceed to implement such things. I hadn't intended to make a case against standardization, but only against restrictive choices in standardization. Frankly, I'm not smart enough to know how to avoid making such choices.

PATRICK: Dick mentioned earlier that you pay a price for generality. We have indicated that there are some people who have two different machines, either physically back to back, or logically back to back across the country who have made such choices in order to handle the situation. All the people who don't have the problem also have to pay the price for generality if the manufacturers are going to provide a translator that contains that generality.

GORDON: I think that Clippinger's comment reinforced things that Armer and I said earlier about standardizing the wrong

things. What you said, Dick, about the FACT compiler and the amount in it devoted to editing, applies also to COBOL and Commercial Translator and similar languages. All of them have to provide for horrendous formats. Perhaps we would all be better off if we were to focus attention more on the microcosm--on things like formats. This would be preferable to having to provide for anything that anyone is likely to dream up in any installation that might use your compiler.

OPLER: How can you convince the customer, though?

GORDON: I think you can convince him of the value of standardization in modules more readily than you can convince him of standardization in the whole thing. We don't really have standardization in things like COBOL either. Look at the insistence on things like USE, ENTER, and other loopholes, which every customer insists on.

OPLER: It amounts to standard circumvention.

GORDON: You might be able to convince customers of standardization on things like tape format, character sets...

BROMBERG: Tape labelling?

GORDON: Labelling conventions, certainly!

DOBRUSKY: You can if you can show him significant payoffs.

GORDON: You can if you can get to them in time.

PATRICK: Galler indicated that they had achieved some measure of upward compatibility in language. He indicated that this should be done at some reasonable cost if you're going to change languages. But if you're talking about files

and file conversion, this is the sort of place where character sets will hit you between the eyes. If I have files created on an RCA machine using one character set and sort order, I can convert it to a Honeywell computer (I picked these two at random because I'm pretty sure they're incompatible in every respect)--sure I can convert from a 6-bit code of one manufacturer to a 6-bit code of any other manufacturer but it isn't only once. If I'm going to maintain a file in Sacremento, in Oklahoma City, and in Maine, I'll have to be converting both ways all the time. I think this is a very serious problem and one that tends to get glossed over. We find people tending to say, "Let's COBOL everything." You can't just COBOL everything unless you get some of these foundations under it.

BROMBERG: Or standardize on one machine--like ours.

*: You mean standardize in the small area.

GRUENBERGER: That's one of the suggestions we made a while ago that we standardize machines by problem area rather than picking them by roulette.

McCRACKEN: I'd like to ask the assembled implementers a question. I talked to the man in charge of software construction for a certain machine. He has 50 people working for him (give or take one or two). I asked him what kind of people these are. He said that he had 6 group leaders who had been in the programming business for three or four years. For the other 44 the average experience is under one year. At the time I was talking to him this group had just

finished, say, COBOL for some machine. I wonder how much of our trouble these days is simply due to the fact that we haven't grown up yet.

GALLER: Dan asked the question but the answer is implicit in it.

CLIPPINGER: The answer was supposed to be a number like 27, or 38.

GALLER: I simply wanted to point out that I know our example well using MAD. The language isn't that different from FORTRAN. We studied what was wrong with FORTRAN in terms of compilation times and so forth. We learned a great deal and we came up with a translator that takes 16,000 instructions instead of 60,000. It compiles on our machine ten times as fast. We have only three people doing it--maybe that's part of the answer.

McCRACKEN: It's a big advantage.

GALLER: The three people who did it were experienced--they knew what they were doing.

GRUENBERGER: You have design control.

GALLER: I don't think the result has to be as bad as some of these speakers here have made out. I'm aware that we can't apply really experienced people to everything. But where it's important I think we must concentrate such people. You are certain to lose when you have 44 people who don't know what they are doing.

McCRACKEN: Look, I asked this guy, "Would you be better off with three people?" And he said, "Of course I would, but

I can't get them." He said, "There is nothing I can do but let them grow up."

LITTLE: Or do it with the six group leaders. How about that?

ARMER: I'd be interested in asking Cheatham how many man years you estimate the task you mentioned will take and what kind of people are going to be doing it.

CHEATHAM: Which task is that?

ARMER: The one where you are writing several compilers for several different machines.

CHEATHAM: Well, for one thing, I never put more than two or three people on one compiler.

GRUENBERGER: What kind of people?

CHEATHAM: The average man, for example, has 6-8 years experience.

LITTLE: You're talking about implementing compilers. I'd like to ask Jerry their experience in implementing systems.

KOORY: What do you mean by systems?

PATRICK: Come on, you can't avoid the question like that.

KOORY: No, but it was worth a try. Do you mean a programming system?

LITTLE: I'm talking about doing a job for a customer as opposed to doing a compiler.

* : What we used to call an application, way back when?

LITTLE: Let me put it this way. How many programmers do you have, what is their level of experience, and what are

you trying to do with them?

KOORY: I'm still somewhat stymied. Are you asking how many people we apply in building a compiler?

LITTLE: I'm trying to get away from compilers and talk about an application.

KOORY: You're asking, for example, how many people do we apply to building a Damage Assessment Model for the DOD?

LITTLE: Yes.

KOORY: Well, of course, this is a function primarily of what we consider to be the requirements of the system. We have just finished a Damage Assessment Model which will operate on the 1604. It was written in JOVIAL. For the actual writing and check-out we used on the order of 25 programmers, as I recall.

LITTLE: Do you have any idea of the level of these 25 programmers?

GORDON: Before or after?

PATRICK: And which way does it go?

KOORY: I would say on the average between one and two years. We are fortunate enough to have four or five of them who each had four or five years of experience. We have had a fair enough number of brand new folks, you might say brought in off the street (that is, just out of college). We had to train this latter group.

* : I hope not out of Bernie's college.

KOORY: The table shows the results obtained when we used an early version of the JOVIAL Compiler for the 1604 in

producing two program systems for a customer. There are a total of 27 programs in the two systems. It should be remembered that the figure does not include program environment (internal data storage) in the statistics shown, only operating instructions.

Program No.	JOVIAL (J) Statements	Generated (G) Instructions*	Ratio (R) G/J	Deviation ± Mean Ratio
1	116	842	7.25	.02
2	205	1340	6.54	.69
3	442	2792	6.32	.91
4	221	1586	7.18	.05
5	242	2400	9.92	2.69
6	105	641	6.10	1.13
7	133	1231	9.27	2.04
8	189	1645	8.71	1.48
9	174	1050	6.04	1.19
10	874	4448	5.09	2.14
11	590	3002	5.09	2.14
12	200	1771	8.85	1.62
13	348	1907	5.48	1.75
14	256	1212	4.74	2.49
15	556	3175	5.70	1.53
16	74	751	10.01	2.78
17	146	1173	8.04	.81
18	142	1088	7.67	.44
19	338	2396	7.09	.14
20	406	2181	5.37	1.86
21	265	2531	9.74	2.51
22	954	5363	5.52	1.71
23	173	1491	8.62	1.39
24	673	3557	5.28	1.95
25	112	1048	9.44	2.21
26	200	1952	9.76	2.53
27	2000	12,688	6.34	.89
	10,134	65,261	195.16	41.09

$\frac{195.16}{27}$ = 7.23 (mean ratio of Generated Instructions to JOVIAL Statements).

$\frac{41.09}{27}$ = 1.52 (mean deviation).

*Program size (G) differs in some cases from the full program size since internal tables are excluded as JOVIAL generated instructions.

Table 1

BROMBERG: I don't know if I'm speaking to McCracken's question or not, but is not this vast number of inexperienced programmers hurting us? I have concluded that it is inevitable that we are always going to have such numbers of inexperienced programmers. When we deal with language implementation for new computers I think the first thing we should do is turn out a compiler and then perhaps a year later turn out a good assembler. I reached this conclusion just because of this problem that Dan brought up. Those customers who are new to the machine are quite similar to those implementers who are new to the business and to that particular machine. Only through use and experience is the customer going to get down to the measure of the efficient use of the machine. At that time they should be given an assembler which allows them to make this efficient use.

McCRACKEN: Leave us just pray that the customers just don't get so furious at the compiler they receive by that process that they give up on COBOL.

PATRICK: Yes, I seem to remember the time when a person would polish like crazy if he was doing utility work, because you would say to yourself, "Gee, if I can save three instructions here, that will be three instructions off everyone's use when they go to use this particular sine routine." What happened to that philosophy? Seems like we're kind of galloping into second shift rental with the philosophy you were just mentioning. We're saying, "It only takes an extra

10,000 instructions so we get into second shift rental in the third month and who cares?"

BROMBERG: It seems to me that the important thing today with new machines initially is to get the application on the air. Get it done and then star' worrying about the techniques for improving it.

PATRICK: It's not clear that you get it done with raw people.

OPLER: I think we ought to have the opinion bag held up permanently now.

I can't see this approach at all.

When you write an application and you make a slight error (e.g., your loop is one instruction too long) you have hurt that application only. When you make the same sort of error in writing a compiler everyone who uses that compiler gets hurt. I feel very strongly about this. When you're writing a compiler, you must have your best people on it exerting their best efforts to squeeze everything out of it they can.

GORDON: This sort of thing should make very happy those people who argue that you can put up with a little inefficiency. We hear a lot from them. What Bromberg said is a good answer to these guys who say, "We want it yesterday and we're willing to put up with a little inefficiency to get it." Howard's approach would give it to them.

* : It might give them a little more than they would want if they follow Howard's suggestion.

MacKENZIE: There's no real basis for comparing the efforts or experience levels. I don't think there is any real argument on this because you have to achieve design control one way or the other. You had best do this with a small experienced group. We have found that it is one thing to do a good programming system with a small group; it's another thing to properly promote it, if you will, to be able to support it at the installation level. It's at these stages, I think, that you have to start applying considerably more labor to these tasks than the compiler writers themselves ever recognized required.

In talking to people about the overall jobs it's important to recognize what the basic responsibility of the group that you call compiler writers is. The outsider tends to regard it simply as the job of coding the processor to go with the programming system. Quite frequently the job requires a great deal more.

PATRICK: Howard, do you want to protect yourself?

BROMBERG: I just don't understand why there seems to be so much agreement on the "fact" that design control depends on the smallest possible number of people.

McCRACKEN: You're making a virtue out of your vices now.

BROMBERG: Clearly, if you have 50 people on a design effort not all 50 are involved in design. Only 4 people will be involved in the design or even only 2. The other people

are going to take the analysis, that is the actual design, and interpret it. The four, who are then the design merchants, are going to be standing over their shoulders. They will make sure that the proper design control is exercised. I don't think design control has anything to do with the number of people you have in the implementing body.

GRUENBERGER: That s wishful thinking.

PATRICK: Isn't that exactly what Opler was talking about when he was speaking of the 200 little, tiny, but very important, decisions? You're not making those decisions with the 4 guys, you're making them with the 46.

BROMBERG: No sir, you're doing it with the 4 guys. The only difference between a big sweat-house operation like ours and the operation at Tom's shop is that his guys are doing everything. Our guys have a little bit more leisure.

GALLER: You don't really know when they are making those decisions. They may not have enough sense to come and ask you about these decisions. If you're there at the time and they happen to think of it, it's O.K.

BROMBERG: This is the problem of implementation follow-up.

MacKENZIE: You seem to have implied that you can distinguish a point in time when the design is finished and the implementation effort is ready to be started. I wish we could sometimes.

GRUENBERGER: All you've got to do is look at one of these languages and you can just about measure the number of

humps on the camel that made it.

BROMBERG: That's fine. I agree when you're talking about language design. But when you're talking about a processor design it's a different thing.

CLIPPINGER: Let's pursue that point. Let's take FACT. There's a language that is obviously very complex. If you're going to look at it and start measuring the number of humps in the camel I think you're going to tell me that there were a lot of people involved in it. The actual number of people isn't more than three or four.

* : But how many humps did they have?

GRUENBERGER: How many people implemented it? How many people actually wrote instructions to create FACT? These people had to make decisions. Each one grew a little hump.

CLIPPINGER: There is certainly some truth in that. I think you tend to exaggerate it though.

GALLER: Let me give another example. Over the last summer we rewrote our entire system to go from the 704 to the 709. We put a group of eight people on it. Eight good people, not the type of the 44 we were mentioning, but eight good people. We had lots of discussions about the specifications and so forth. You can always discuss down to a certain level. Each of these people went away and did a job. The resulting system is real nice; it's efficient and it's running beautifully. I'm <u>still</u> finding out things that these guys did; each in his own part. I know, overall, what the system

is doing and it's doing what it's supposed to do. They came to me with lots of questions during the implementation. They'd ask me if it was O.K. to do a certain thing and I'd evaluate their questions in terms of the overall problem. But there were lots of things they did where they didn't ask me and I have no idea what they did. Some of them I agree with; some of them I don't. They are just now coming to light. Neither I nor a group of any size could have possibly overseen what these eight men were doing. Since I was the supervisor on that particular project it was a ratio of 1 to 8. How many would we have had to have supervising these people to really keep control? And how many would we have had to have on top of them?

GORDON: That's what the other 44 could do.

I'd like to spell out as an axiom that no programming language is fully defined until there exists at least one compiler for it.

CLIPPINGER: I would go further than that. It's not defined until it's defunct.

GORDON: I'm not ready to go that far.

MacKENZIE: One possible way around this problem that was mentioned on design and implementation is to find sufficient means to convey the design to the implementers. In many cases if you can do this you can eliminate the implementers to a great extent. This comes back then to some virtue, if you will, of an approach to standardization. For example,

writing a processor in its own language certainly tends to minimize the number of decisions that the actual implementers might otherwise be about to make.

PATRICK: I don't understand how that comes about.

MacKENZIE: I made the remark before--we were talking about the problem of design at one end and applying that design to large bodies of implementers at the other end--that people deviate from the designer's intent because it wasn't quite clear what the designer's intentions were, quite frequently. There are a lot of other reasons why people deviate too. Perhaps they didn't agree with the designer. A conveyance of the design specifications is an extremely important thing. Describing a processor in its own language is one way of minimizing, I think, the prerogatives that the implementers would otherwise exercise if for no other reason than it tends to reduce the scope of the implementation effort.

PATRICK: I still don't see how this helps.

MacKENZIE: What I was saying clearly implies that the designers are doing most of the implementation in expressing their design.

GORDON: Maybe there is no clear-cut sharp line between the design and the implementation. We saw one example of this in July of 1960 when we published a manual for Commercial Translator language. As late as a year after this thing was published we were discovering what we meant by some of the

things in the book as we were getting around to them. The book described these things in general terms, and gave an idea of the kind of thing we would have. But what would happen specifically under the conditions of compiling we hadn't gotten around to yet. As these things came up for implementation they became pinned down. I don't think that you can say that we will design up to a certain point and then throw a switch and be implementing.

CLIPPINGER: What is the subject of this conference?

PATRICK: The pros and cons of common languages. And what we're after is enlightenment.

CLIPPINGER: I'm not sure we've been talking about that subject.

PATRICK: Perhaps we've digressed a little bit to more than a level of detail. So it might be appropriate to review what we've covered this morning.

We started out talking about visceral feelings. These were things that appealed to us, like why common languages were good and why they were bad. These are God and Mother categories on a high plane. Along the way we uncovered just a few facts. We have made some statements that such languages are not completely machine-independent. As of the present state-of-the-art, we don't seem to know exactly how to do this. In the implementation stage you made them, perhaps, a little more machine-dependent in order to get some efficiency. We pointed out that you pay for generality. We noted that it

was great to have a language that you can translate across all machines and have it efficient on every machine. I think it is our concensus that we don't quite know how to do this just yet.

The last topic we were on, I think, was whether it was better to have a high quality small staff or a young mob. With all deference to Bromberg, I think he was talking about a pyramid of a staff, with a genius at the top and levels of priests and sub-priests down to the machine clerks at the bottom. It's rather difficult to administer such a staff because the design process extends all the way down to the key-punch stage.

L U N C H B R E A K

ARMER: Let me ask a question. There have been rumors about the Navy going the JOVIAL route. If this is true we may find, two years from now, we are in the same boat we were with FORTRAN some time ago--nobody likes it but the investment was just so great that we had to continue going that way. Might we recommend that there be less emphasis on JOVIAL and more on say, NELIAC, so that we can try something else; so that we don't have this tremendous commitment to one language two years from now?

* : How do you get out of that box? The only thing you can recommend is: don't standardize on JOVIAL--get spread so thin that there will not be so much resistance to going down one particular road later--but so what?

OPLER: To this point I think the really important thing is the question of the time scale.

In preparation for this meeting I sat down and wrote out some of the arguments against standardization. I decided that I would wear my anti-standards hat this time. (I have also a pro-standards hat.) Eventually the arguments boil down to two classes. One is the set of arguments against programming language standardization at this time. The other is the set of arguments against programming language standardization at any time. I think the more interesting set is the arguments against standardization at this time. Just to show that I was trying to be objective I sat down last night and wrote down all the counter-arguments against the arguments against standardization.

PATRICK: I think it would be appropriate to list these arguments on the board.

KOORY: I'd like to ask if we're listing arguments against standardization from the point of view of the DOD or from the point of view of the rest of the computing world.

OPLER: Actually, these are from the point of view of the rest of the computing world although I have been doing a lot of thinking about the problems with reference to Command and Control.

The arguments against standardization now are as follows:

1. Programming languages are changing too much.
2. Programming languages have not developed sufficiently.
3. Promising alternatives are just now appearing.

The arguments against standardization at any time are as follows:

1. Administration is too time-consuming.
2. Poor past experience.
3. Adverse effect on computer progress.
4. Specialized languages are more efficient.
5. Programming languages are not the right level.
6. Language standardization is only a part of the solution.

Figure 3

ARMER: I'd like to hear Ascher elaborate on that point: "Alternatives are just now appearing." I'd like to hear him enumerate one or two.

OPLER: I think this is one of the most serious arguments against standardization right now. It seems that we're just at the beginning of a new period.

In the past it has been believed that the preparation of a compiler to translate from a source language to an object code has been a tremendous job--one that can't be done every day, that demands man-years of effort. New tools, such as syntax direction and table direction and the type of compilers that can be built on hierarchies and list structures, give to the future a freedom that we haven't had in the past. We will have freedom to agree on something radically new, implement it, test it, and if we don't like it, throw it out and iterate again. In the past we have had 20 to 50 man-years devoted to a compiler to the point where once you have created it you have too much of an effort expended--too much inertia to overcome--to try to change it. I am reminded of a remark by the late Dudley Buck, speaking about microminiaturization and printed circuits when he said that someday we may be able to write our computer instead of writing the programs that we want. With syntax direction, if you don't like the language you've developed today you can write a new one tomorrow. So one promising alternative is the release we now have from this vast implementation effort.

LITTLE: From the user's standpoint I hear you digging my grave. Right now they change them to the point where it's difficult to keep going. You see I think that testing of a

language is done when you implement real jobs. If I'm going to have to implement real jobs some of which are obviously going to take longer than it takes you to write a new compiler, somehow our time phasing will never be in synch.

OPLER: But Jack, I'm talking about language development.

PATRICK: The kind of stuff that Galler is doing, not the kind of stuff that is done out in the field.

LITTLE: But I have to use something everyday to do the job with.

OPLER: I understand that. I'm just thinking that in 1975 we will look back and feel that in 1965 we were just beginning to find out what the structure of these languages was. 1965 marked the time when the second big round of these experiments began to end. I feel that we have just begun to climb but have scarcely gotten off the toe of the curve. I hardly think that we can say now that we have reached the plateau; that we can say that we know enough now to set up our first standards.

My feeling is that we are at the beginning of a very exciting period of development. If we standardize now, we will be sorry. I do not think we can now look back to a long enough period in which various ideas have been tested and rejected.

LITTLE: I agree, Ascher, that we are probably on the verge of some very exciting times in developing the languages, but we are also on the verge of some very exciting times in

implementing very large jobs, like Command and Control systems. If your argument doesn't carry over in some proportion to enable us to do these jobs either over or better now, we're in some sort of trouble.

DOBRUSKY: It seems to me we're confusing the difference between language and the implementation of that language. We've talked about the techniques of things like table handling, list structures, and the syntax of the language. The language itself is not reflected necessarily in the way we implement it. Anyone capable of writing a compiler can take any language that exists today and write an implementer for it that will take advantage of one measure of efficiency. With various implementations we will find the best means of doing this with the help of the universities and of various groups in private industry. I think the direction of a language for describing problems had better be solidified somewhere. We must cut down continued proliferation of different languages. We still don't have a measure of their effectiveness yet.

OPLER: Let's consider all the effort that has gone into NELIAC and JOVIAL, for example. Supposing tomorrow someone comes up with a radically new language. Suppose this new language requires the expenditure of 30 to 50 man-years of effort to get it to the point where we can put it to use and see if it's any good. If we can write a syntax table for this new language so that we can give it to somebody and have a compiler in a few days, then we have given people more freedom

to design languages. We are not then restricted to a small set of languages like the ones we have now.

DOBRUSKY: Ascher, I agree with that approach but I don't think we're smart enough to write such a processor yet. I don't think we can achieve the various measures of efficiency that have been described here such as fast compile time, efficient object code, easy training, and so on.

GRUENBERGER: You guys aren't arguing.

OPLER: I agree, I don't think we really are arguing. I am simply saying that I think it would be premature to standardize because we are just now getting into our hands new tools to work the area. I don't think it has been proven that programming language standardization is the only point of standardization. Perhaps we should consider procedure standardization, problem definition standardization, description standardization, or data standardization.

McCRACKEN: Perhaps programming language standardization is the only one we know how to attack now.

PATRICK: Maybe we're not attacking one where a problem exists. Maybe the problem isn't in the coding.

ARMER: You don't think that we know how to standardize on files, on labelling, on format, on character sets, and things like that?

McCRACKEN: All right, take files, for instance. I don't know how we could standardize there until we have standardized on character sets first.

ARMER: O.K., so we must standardize character sets first.

McCRACKEN: O.K., give them your speech, Dick.

CLIPPINGER: You mean the work that is going on in character sets? Well, X-3.2 has a proposal for an American standard on character sets. I thought you would all be aware of this.

McCRACKEN: It's up for a vote at the present moment.

CLIPPINGER: It's reached the point where X-3 has had it in their hands for a while and it's up for a vote. The votes are supposed to be in by the end of June.

PATRICK: After they vote will the rest of us get to see it?

McCRACKEN: It was published somewhere about a month ago.

CLIPPINGER: There are users groups in X-3 including JUG. Harry Cantrell, for example, is involved here.

PHILLIPS: This includes such groups as the Air Transport Association, ABA, and National Retail Merchants Association. There are nine groups all together.

GALLER: What would be the effect if this resolution goes through for a standard character set?

CLIPPINGER: I would guess that the effects are going to be extremely extensive. You can use your judgment on this as well as I can. IBM, for example, has a group set up to study the effects and see what they will be on IBM. I'm sure that when they complete their study they still won't know the extent of the effects. Look at it yourself. There

are 128 characters in this set, with a subset of 64 standard characters. I'm sure you'd like to see this standard reflected in such things as keypunches. The letters of the alphabet, for example, are a connected set among these and they are not in the current keypunches, nor on the 407 printers. The collating sequence of every machine will be affected. The appropriate packages for minimizing the cost of the inside of the computer would probably be affected. It seems to me that every aspect of computer hardware will be affected. Now, of course, a standard is never compulsory and it's up to each person to decide to what extent he wants to go along with it. Clearly, for example, IBM is not going to obsolete all its keypunches overnight, nor all of its printers. Neither are the rest of us. You can see, though, that there would be economic pressure on each manufacturer to move in this direction so that his costs would go down as a result of accepting this standard. Who can say what the total effect is going to be? There will be some effects which will increase costs and some which will decrease costs for the manufacturer and the user and what the net result will be I'm not sure. Personally, I think the net result will be to decrease costs. This is just an intuitive feeling.

To those of you to whom this is complete news, let me say that X-3.2 has worked a couple of years at this problem and worked very hard. They have explained very carefully how they went about arriving at the conclusions that they arrived

at. There is a 20- or 30-page document--pretty well put together--that you will all want to read.

PHILLIPS: I can furnish this document to anyone who is interested. I also now have found a list of the groups involved in setting up this standard. There is LOMA, JUG, the American Bankers Association, the American Petroleum Industries Association, the American Gas Association and Electrical Industries, General Services Administration.

PATRICK: GSA is an important one.

PHILLIPS: Yes, GSA is listed among the users and the Department of Defense is listed under the general interest groups. Among the manufacturers we have IBM, RCA, NCR, Monroe, Remington Rand, Minneapolis-Honeywell, Pitney-Bowes, Standard Register, Burroughs, Royal McBee,...

GORDON: What about NMAA?

PHILLIPS: They should be here somewhere. Let me go on. In the general interest groups, we have ACM, Department of Defense, The Engineer's Joint Council, the Telephone Group, AIEE, ACM, ERA, IRE, and NMAA. The American Management Association is in the general interest group and also EIA, the Electronic Industries Association. There are ten general interest members, nine users groups, and ten manufacturers.

CLIPPINGER: If you're interested, this character set has provision in it for the letters of the alphabet and the ten decimal digits, of course. The usual special characters that we're familiar with, control characters (such things as

carriage return and carriage shift on a typewriter), data delimiters (end of field, end of group), and a character for escape which is an important one in case you want to type in an entirely different set of characters (a group like the weather bureau might like to do this). There are other control characters such as those used for wire transmission ("where are you") and so forth.

Some thought has been given to the problems of international communications; the substitution of alphabets for example. Members of X-3.2 have travelled through Europe and held discussions with other national standardizing bodies. They have explored the possibility of moving toward international agreement. There are people in the United States who are not interested in any standard coded character sets unless they can be assured that it will also be an international standard. Of course an international standard is much more difficult to achieve. (In general, in the world there are about 2,000 national standards in a country like the United States and only about 100 international standards.) I'm sure if you examine the character set that is being proposed you can imagine some of the implications yourself.

In preparing for a meeting in Stockholm of the Programming Languages Standardization Committee, we put an item on the agenda to consider the implications on programming languages of a standard coded character set. Bob Bemer is summarizing the results of such activities in a note.

The character set proposed by X-3.2 contains all of the characters used in COBOL and many of those currently used in ALGOL. It couldn't include all of those that you might like to use in ALGOL. That would probably take more than 128 alone, but obviously no one now using ALGOL needs nearly this number anyway. You might, however, like to take advantage of this escape mechanism whereby you can use any other character sets you would like. In the FACT language we provide for editing information coming in from paper tape. As you know, people have a tendency to use 5-, 6-, 7-, or 8-level paper tape; they like to indicate the end of a field with an end of field character, for example. FACT recognized all such characters, provided that you define them to FACT by means of tables.

Well, I've tried to list some of the implications to programming languages of a standard coded character set. It's a very complex problem.

PHILLIPS: There's something I would like to add to that. In the X-3.2 area we have an example of premature standardization. Several years ago the Department of Defense, in the absence of anything better, adopted what is now called the Field Data Code. The Army led this off. I contend that this standardization was premature because they did not bring into the discussion as far as I know, representatives from the data processing community as well as communications people. It is primarily a communications code. They made no provision

for expansion of the alphabet beyond the English alphabet. The order in which the characters were arranged in the code did not lend itself well to sort and collate operations. And yet it was considered as a principal candidate for the character string by X-3.2. Incidentally, the Navy and the Air Force have since adopted it; so it is now standard for the military. I'm not sure whether or not it has been adopted by civilian agencies of the government.

PATRICK: NATO is currently discussing it.

PHILLIPS: Here is an instance where the Federal Government--the biggest user in this field--adopted a standard several years ago and yet it may be cast aside by the X-3.2 group. That group thoroughly considered it before designing the present code we were talking about.

CLIPPINGER: Since there seems to be a large lack of information about what's actually going on in standardization it might be worthwhile mentioning one or two other things.

The X-3.1 group is trying to decide on a character set for optical character reading. This could be the first standard in that particular area. The current status of that effort is that they have tentatively decided on a set of 16 characters in three different sizes. (There was much debate as to whether they could get by with one size or whether they would need a set of sizes.) The ABA standard for MICR will also be proposed, of course, through X-3.7.

X-3.6 is working on a set of flow chart symbols. There are also people in X-3.4 (programming languages) who are

working on flow charting symbols. There is more to it than just agreeing on symbols.

PHILLIPS: X-3.6 also has problem description and analysis.

CLIPPINGER: X-3.5 is working on a glossary in the data processing field.

X-3.3 concerns itself with communication. It works in close conjunction with the EIA committee. They have a proposal in the mill for a set of standards on transmission frequencies. They are also considering standards for input/output media (cards, paper tape, and so forth).

GALLER: You mentioned the MICR standards. When I saw it they were concerned only with standard shapes for the digits. Do they have alphabetic characters in there too? I was astounded when I found that they showed such little foresight in allowing only for decimal digits.

CLIPPINGER: I wouldn't want to defend that but you have to recognize that the ability to recognize 24 or 36 characters with little chance of error (rather than the 16 they're working on) is a difficult engineering problem.

BROMBERG: To do it later is even more difficult.

GALLER: After you set standards to provide sensitivity to distinguish between 16 characters, then to later extend those standards to provide sensitivity to distinguish between 64 characters is a much worse problem. It's almost impossible.

GRUENBERGER: All you have to do is specify that the new system must be compatible with the old.

get some controversy going on this.

PATRICK: It sounds to me like from what Dick Clippinger and Charlie Phillips have said that they are laying what looks like a superb foundation to build on. It looks to me like a strong argument for Opler's case of waiting a while.

BAGLEY: How long do you wait before you pick one in order to get your current work done? I feel encouraged by this SDC report. They happen to have picked two years. I'm not that much of an optimist. I would still like to see the kind of effort they suggest; namely, get a bunch of brains together like was done in the COBOL effort.

* : They picked six months.

CLIPPINGER: I'm sure it would be desirable at some time in the future to have programming languages that are better than the ones we have now. On the other hand all the manufacturers are faced with the problem of providing COBOL for their customers. There are questions of interpretation that arise and it costs us money. It costs us money when we can't get an answer and it costs us money later on when we can get an answer. I think a good case can be made, particularly in the case of COBOL, for accelerating the process by which the language becomes defined. Just the fact that 15 manufacturers are implementing COBOL on 35 different machines makes a case, it seems to me, for declaring it to be standard at an early date. I think we need a period of use, for the languages we now have will provide a criterion for the

GALLER: This is some constraint.

GRUENBERGER: You bettcha it is!

PATRICK: This is the same problem that the military has with their Field Data codes. Since they adopted Field Data we actually have Field Data constructed into the hardware in these big communications systems. If they had said, "Stay loose boys, we're not sure which one we're going to adopt," it then could have been implemented several different ways at no more additional cost.

In my opinion this is a clear-cut case of suboptimization. It's like the language designers designing a language that is easy to write. That is obviously suboptimization, because they forgot about the damn user. In this case they sub-optimized on something for communications purposes without any thought of what you want to communicate. If you want to order some Air Force parts from Oklahoma City, you're going into a data processing system.

PHILLIPS: I think there's a difference here. If you adopt something as a military standard as they have with the Field Data Code, I think that all three of the military departments are required to follow that standard. Their alternative in this case (referring to X-3.2) is to adopt it as an alternative standard. If there is an alternate standard then you have the option of going with whichever one is best suited to your needs.

ARMER: I still haven't heard anyone say, "We ought to be going this way instead of the way we're going." I'd like to

improvements in future languages.

LITTLE: We need the use but we also ought to investigate what these criteria should be. For example, FORTRAN has been in use for quite a long time but if you try to go and get any statistics on how it operates you bump up against a pretty cold, hard wall. Usage alone doesn't do the job.

GORDON: COBOL is not a standard, de facto or anything else; just the fact that you have N manufacturers implementing something, on M machines, all called by the same name does not make it a standard. You know, and I know, Dick, that there are no standard programming languages in the commercial area today. One COBOL looks as much like another COBOL as FACT and Commercial Translator look the same.

CLIPPINGER: I agree with you Barry, of course. What I meant was that the notion of COBOL has achieved a kind of acceptance which is extremely broad. There is something standard about it, although it is certainly not the language itself.

GORDON: The name is about the only thing that is standard.

KOORY: It's not that bad.

BROMBERG: I would much rather have the job of taking a Honeywell COBOL program and converting it so that it would be acceptable to an RCA COBOL translator than I would to take a COMTRAN program and convert it to any other machine.

GORDON: Even leaving Honeywell out of it, is it possible to take a COBOL program written for the RCA 501 and convert it to the 601 without a major rewrite?

BROMBERG: Of course it is. It is even easier because of the family relationship which can be accomodated automatically by the object compiler.

GRUENBERGER: I'd like to address a question to the manufacturers' representatives who are here. If you, the manufacturers, could have your druthers and the DOD said tomorrow morning, "KLUDGETRAN is it, by golly we're frozen!" (and let's assume that they could define KLUDGETRAN)--that's the language for Command and Control--would that make you happy? Or would you like to see this decision put off another five years? Define it any way you want, but answer the question.

BROMBERG: If you have a language specification to which a number of people agree and they agree that this language would perform the job and you have one big fat user who says, "Yes, I'm going to use it," and if you have a strong authoritative body that is going to handle the interpretation, modification, and extension of the language, I think that we, as a manufacturer, would be glad.

GORDON: Now I know, Howard, why you insisted on those compound "IF" statements, but I agree with you anyhow.

My personal opinion is that if anyone were to come up with a proposed standard in this area tomorrow morning it would be premature by several years, and would be, in the long run, a bad thing for the industry.

GRUENBERGER: Then you'd be unhappy.

PATRICK: This is in the business area you're talking about Barry, isn't it?

GORDON: No, he said Command and Control. The business language area is one in which we might be ready to standardize within two years, but in Command and Control I think it's farther away.

PATRICK: Could we list on the blackboard the reasons why you feel that way, and see if we could get some agreement on it; that is, why you feel we shouldn't standardize now on a Command and Control language?

GORDON: Because we don't have any Command and Control languages yet which are generally felt to be worth standardizing on. It's simply that a Command and Control language has not yet evolved; it's just that simple. I'm of the evolution school. Perhaps the noun "standard" or the verb "standardize" needs some definition. To me the verb "standardize" means to declare or recognize as a standard. When I think of standardizing I do not think of creating a standard.

GRUENBERGER: All right, suppose we put it this way. We have a KLUDGETRAN definition and we have already implemented it on five machines. Now the proposal is made that this language we have which is a real hot-dog language--it does everything (oh, there are a few things it doesn't do, like handle algebraic statements, or loops, and so forth)--be made the standard Command and Control language.

GORDON: Then I'd say fine, let's publish the darn thing and let's see people jump to use it because it's so great. Then a year from now when 85% of the industry has

embraced it, it is obviously a standard and let's say so then.

BROMBERG: Should we carry over the same exact remark that you just made, Barry, into the current real practical world as it now exists around COBOL?

GORDON: No, we can't.

BROMBERG: Why not?

GORDON: Because of the Defense Department.

* : There's not a free choice there.

PHILLIPS: They're the only ones who have a basic use for it. They're the only ones who would use Command and Control.

ARMER: But Howard put it into a different context, referring to COBOL. There's another way for the world to go. The DOD could rescind their statement about COBOL--that statement about they're not going to order a machine unless there is a COBOL translator for it.

PHILLIPS: You're going to have to read that thing, Paul; it doesn't say that.

PATRICK: Yes, but it works that way.

GORDON: Crabgrass doesn't say that it's going to take over the lawn, Charlie; it just works that way.

ARMER: You understand I'm not arguing this point, Charlie, I'm just trying to get some discussion because I think it's rather implicit in some of the things that are being said.

PHILLIPS: The statement says that you will purchase a machine that has a COBOL compiler available unless there are

reasons why you don't need it.

PATRICK: Who judges? Has anyone done it?

MacKENZIE: I know what the directive says and I think I understand the reasons behind it, but is its intent carried forward into the invitations to bid? I don't believe it is.

PHILLIPS: All three of the departments are going through the problem now of deciding how and where to make their equipment selections. This has been going on for some time. There isn't a lot of selection that is done at the local level anymore. In fact, it hasn't been for some time. The ones who will be making equipment selections at the various echelons are fully aware of the fact that they don't have to conform to this directive if they don't have reasons to use COBOL.

CLIPPINGER: What you're saying expresses the psychology of the DOD user but from the psychology of the manufacturer--he has salesmen out there trying to sell his machines and the people he's trying to sell them to are saying, "Look at the GSA contract." Now can you choose freely whether you are going to use COBOL or not? The answer, of course, is pretty clear. You're going to do COBOL. It doesn't matter what it costs you, you have to do COBOL so that you get a fair chance to market your equipment.

* : There were a lot of users too, at the last SHARE meeting and the last Commercial Translator meeting who said that they were going to go COBOL simply because of the DOD requirement.

ARMER: It seems pretty obvious to us that IBM is going to abandon Commercial Translator because of COBOL. I've never heard them say so explicitly, but you can judge by their actions.

GORDON: There were people who were unhappy about this in IBM but there wasn't really any choice.

Speaking of Commercial Translator reminds me of something that I found very interesting. One of the things we were asked to read before this meeting, was Joe Wegstein's article on ALGOL in the September 1961, Datamation. Let me quote from it:

"... have led to the development of numerous artificial computer languages such as: FORTRAN, ALTAC, IT, FLOWMATIC, COBOL, ALGOL, LISP, COMIT, IPL, JOVIAL, MAD, and NELIAC."

You know, Dick, we might just as well not have bothered. The two independent research activities in the Commercial languages have sort of been written out. They have been "new-thunk" out of existence.

PATRICK: It does seem as though DOD is a wet blanket here, whether you really intended to be or not.

PHILLIPS: It was intended to support COBOL, let's not be coy about it. You try to face these things positively. If you're going to support something you do it positively and we were supporting COBOL.

GALLER: I understand that CODASYL has a long range effort also. I remember reading something in Datamation about

decision tables influencing this language. What is the position of COBOL with regard to this long range effort? Is COBOL a subset of what they hope to accomplish? If it isn't, what will the status of COBOL be? What are the attitudes?

CLIPPINGER: The article you read, of course, represents just someone's opinion.

GALLER: Well, whatever happens, whatever they come up with, are they pledged to COBOL or can they go off in new directions? Is the long range group committed to the short range group's COBOL?

PHILLIPS: Not necessarily.

CLIPPINGER: Speaking as a member of the language structures group of CODASYL, there is an article in the current (April) issue of the ACM Communications on "Information Algebra" done by a group of people in CODASYL with no connection with anything else. These are six people who simply found it convenient to go off by themselves and do some work. The work of the systems group is an extension of the notions of Burt Grad on tabular languages. This group is working independently of the COBOL effort and I would guess it would be at least a couple of years before we could properly evaluate their work. It's too early to tell whether it will have any influence with what is going on with COBOL. The COBOL group at the moment is polishing the extensions that they have already decided to add to COBOL (such as a report writer and

sort)--these will go into Extended COBOL 61. They will then turn their attention to clarifying ambiguities in the current language; that is, maintenance.

They will eventually put all this together in what will probably be called COBOL-63. It is most unlikely that there will be any influence by the systems group on that effort. It is my opinion that you won't see any interaction between the systems group and the COBOL group before 1964, and then it will be done by salesmanship on the part of the people in the COBOL group. This is all personal opinion, of course.

GORDON: Dick, is it not true that the systems group in developing these tabular formats have been working along the lines of having the COBOL 61 language be the language used in the tabular format?

CLIPPINGER: Yes.

GORDON: So I would question your saying that they're working independently of each other. They're working independently of the COBOL committee, but they are not independent of COBOL at all. They are merely casting COBOL into tabular form the way it looks now. They are making a few slight improvements like using the word "SET" instead of "COMPUTE" (and other such "radical" advances). In addition, they're recasting COBOL 61 into square form.

PATRICK: This is some of Bernie's upwards compatibility.

CLIPPINGER: But this probably wouldn't see the light of a COBOL proposal--that is, something to be adopted--before 1964, looking at the rate at which these things are adopted.

GORDON: This ties in with the item of frozen progress on our chart. In 1964 we'll still be effectively using the 1960 short range committee language.

PATRICK: That is the six month version.

GORDON: Right.

GALLER: It's not really my idea of upwards compatibility. The upwards idea is that you're free to try something else provided that you can map into it. These people aren't feeling quite that free. As the years go by, and people write more and more programs in COBOL, they're going to feel less free the same way people are in FORTRAN now.

LITTLE: We keep coming back to the distinction between common and standard and I'm not sure we've made the distinction clear. And we keep wandering away from Command and Control languages which interests me. That's all right; I didn't even get a letter. Anyway, supposing we don't come up with standard language for Command and Control, how about a common language?

GRUENBERGER: As far as that goes, I asked a question a little while ago and two of our alert manufacturers answered it, but two of them didn't seem to answer it and I'd still like to hear what they have to say. The two who answered it were poles apart, incidentally.

CLIPPINGER: You won't catch me saying anything now.

MacKENZIE: The reason I haven't answered your question, Fred, is that I'm not familiar with this subject.

GORDON: That didn't keep me from replying and that's a fact.

CLIPPINGER: I'll answer anyway. The general frame of reference in which Honeywell works (and I don't think it's too different from others) is that if something is required because our users need it, and it will help to sell machines, then we'll provide it and we'd like it to cost as little as possible. We'd also like it to be as good as possible. If it is well defined, that will help us to get it done at less cost. You said, among other things Fred, that we assume that it's well defined and here today and the question was, "Would that make us happy?" The fact that it's well defined would definitely make us happy, but I still don't believe your postulates and so my answer doesn't mean much. I don't think such a language will be well defined.

PHILLIPS: Fred, if you can compare today's situation regarding a Command and Control language with what we had in 1959 and 1960 with regard to COBOL, then I would offer this thought. At that particular point in time several manufacturers came to me and said, "Take a firm, strong hand toward COBOL and push it. We'd like to have you." By this they meant that they'd like Defense to take a strong position on COBOL and push it.

* : I'll bet IBM wasn't one of them.

GORDON: I would guess that neither IBM nor Honeywell were among those manufacturers because these are the two

companies that had taken the initiative and put in some work in this area where a need was felt. These two companies saw the need and did something to fill that need along two similar but somewhat different approaches.

CUNNINGHAM: At the risk of going back just a little bit further in history, the Air Force attempted to get the manufacturers interested in developing a common language in 1957, with AIMACO. Only one manufacturer supported it. The others decided to develop their own common languages. So--

GORDON: But I think you're confusing two different things, Joe.

CUNNINGHAM: No, I'm only speaking to your point that IBM and Honeywell weren't interested in what the manufacturers had asked the Defense Department. I'm just going back a little further and pointing out that the Defense Department (the Air Materiel Command) had asked you to work with them in solving this problem long before you had anything but visions of COMTRAN.

CLIPPINGER: Is this discussion advancing the ball in any way?

PATRICK: I think it has one interesting implication. The military didn't know what they were asking for in 1957 and hence misinterpreted a response. We had FLOWMATIC in 1957, which was just developing. This didn't have any of

the facets that were advertised with COBOL. It was not machine-independent, and it did not materially raise the output of the programmer. Then Wright Field rewrote the thing into AIMACO. When COBOL was first launched the AIMACO translators weren't working. I know, because I was working for Clippinger at the time. That was all just sales talk and you guys couldn't even see through it. You had bought a pig in a poke and didn't know it.

GRUENBERGER: It never did work.

CUNNINGHAM: I'd really have to ask someone here who is more familiar with it than I am.

PATRICK: I'd be delighted to discuss it with him too.

PHILLIPS: Bear in mind that I did not identify the level at which I was encouraged.

PATRICK: It's like a bunch of kids with their nose pushed against the glass of a restaurant and they're saying, "Gee, Charlie, do something to get me a ticket." Some of us were inside eating and we didn't need any ticket. Charlie was busy redistributing the wealth and giving everybody on the outside a ticket. The people on the inside were supposed to help him redistribute that wealth. Note that sign on the wall: "You can't put garbage in one end and get fruit salad out the other." I think we could be in the same position with Command and Control languages today.

I'd like to chalk up another fact and you fellows from SDC can feel free to challenge it if you want to.

(Patrick added to the list on the board the phrase "object efficiency is poor.") I'd like to chalk that up for Command and Control.

DOBRUSKY: I don't know what it means so I can't address myself to that.

PATRICK: I mean efficiency compared to what you and I could do coding an assembly language.

DOBRUSKY: The whole job?

* : You couldn't do it.

PATRICK: The part that's important to get the job done, we could. "The object efficiency of the running code is poor. Their jobs are already pushing the machine."

DOBRUSKY: Jerry Koory and his group have just finished a job (that we discussed earlier) with 65,000 instructions. Could you have done it, Jerry, in the allotted time with the level of people you had without a compiler?

KOORY: I'll have to think a bit before I can answer that.

PATRICK: Let's put it this way. Isn't that about the same order of magnitude as the FACT compiler, Dick? He just said 65,000 instructions; didn't you say you had 285,000 in the FACT compiler, Dick?

CLIPPINGER: He's talking about one-address instructions which makes it about half the size of FACT.

GALLER: You're really talking about two different questions. You're changing the measure. One man says the object code efficiency is poor; the other one says you couldn't get it done in that time.

LITTLE: We haven't really given Koory a chance to answer yet.

GORDON: I think he's pretty happy about that situation.

KOORY: While looking over our SAGE experience and comparing our productive rate there with what we've done in the last six months (with the present compiler we have on the 1604) we find that we produce programs faster--from design through systems test--than we could with hand coding. By faster, I'm referring to the rate of checked out instructions per day.

DOBRUSKY: I agree with Galler here; I did not address myself to the efficiency. This is <u>one</u> measure of what a compiler will do for you. Again, you can achieve better efficiency with either a POL or an ML. As I said earlier, if you use the whole power of the language there are some forms that we know indeed will produce more efficient code than if we use parts of the language. If we use a complex FOR statement it does not produce as good machine object as if we break it up and get closer to the machine language itself.

PATRICK: Okay, but Jerry spoke on a different subject from what you're now talking about. Jerry spoke on the subject of whether higher level language would help Command and Control, which doesn't have a thing to do with standards.

GRUENBERGER: Or object efficiency.

ARMER: If they don't help, then there is mighty little reason for standardization. We sure don't want to standardize on some things which results in our doing the job slower and in a poorer fashion.

OPLER: We agree that higher level languages are better for the kind of systems you're talking about than lower level languages. Some say that standardization is better than non-standardization, and draw the conclusion that standardized higher level languages are best for doing the job. What is confusing us is the difference between the gain from higher level languages and that from standard languages.

PATRICK: Jerry said that a higher level language helped him in doing the code that they just delivered.

ARMERDING: That's not really what he said, is it? He said they produced more checked out instructions, by which I think he meant more machine instructions, but he didn't say whether or not that was an efficiently written list of machine instructions.

PATRICK: That's right.

ARMERDING: I can create a compiler than can turn out 500 machine language instructions for every statement written.

PATRICK: Most of them NO-OPS.

ARMERDING: Right. The sort of thing where three hand-written instructions would really do the job to represent each statement. The only way to really test that would be to take two groups of like people and put them side by side and have them both do the same job, one group using JOVIAL and the other group using assembly language.

GALLER: What's the measure of comparison then? How well the object program runs? Or how soon they finish the job? Or how soon they get answers? Or what?

PATRICK: The subject I had up there on the blackboard was object code efficiency.

LITTLE: Or is the real difference you're talking about the ability of the group itself?

GORDON: Let me go off at a slight tangent here. Armer raised the question a couple of times. "What should we be doing?"

Now perhaps I'm on a subject that's irrelevant to the current discussion. But we've been talking about standardization; standardization of character sets and formats and languages among other things. Armer asks what we should be doing and up on the chart there we have subjects like training, level of users, etc. Maybe we should consider standardization of programmer levels. Maybe we should define what a guy ought to know before he calls himself a programmer. Maybe if we establish certain minimum standards of programmer competence

we might then have a lot less trouble with worrying about standards of programming language and object code efficiency. How's that for where we should be going, Paul?

ARMER: That's a pretty good one, Barry.

GALLER: There is too much of a shortage of people right now to enforce standards like that.

GRUENBERGER: It's not getting any better.

PATRICK: As an example, I will predict what our most commonly used language is going to be in the next few years: It's going to be SPS for the 1401. International Bullmoose is delivering eleven 1401's per calendar day, and they have 7,000 or so on back order. SPS is going to be your most popular language.

LITTLE: There's something else that's not clear. There may be a shortage of people but we may be creating it to some extent. To take the example Dan used, we have six good people who are not really very good floating around. Did we really need them?

OPLER: Be careful. Those are the people who will be the compiler experts next year and will be ringing your doorbell.

GRUENBERGER: You know, I've never seen a hot dog language come out yet in the last 14 years--beginning with Mrs. Hopper's A-0 compiler (you'll pardon the expression)--that didn't have tied to it the claim in its brochure that this one will eliminate all programmers. The last one we got was just three days ago. Like all the others, it makes the

same claim for the G-WIZ compiler that this one will eliminate programmers. Managers can now do their own programming; engineers can do their own programming, etc. As always, the claim seems to be made that programmers are not needed anymore.

DOBRUSKY: I'll take exception with you. The JOVIAL brochure says, "This is for programmers." You're going to need more of them.

ARMERDING: Okay. But to whom are these ads addressed? When computer manufacturers put out ads like that, are they speaking to programmers? Of course not; these ads are not run in the Communications or in Datamation. They are run in Business Week and Time and in the Wall Street Journal. Whom are they talking to?

PATRICK: They are talking to the same guys that didn't understand IBM's reluctance to get on the FLOWMATIC bandwagon. They're talking to the guys who buy the machines.

DOBRUSKY: I'd like to get back to Bob Patrick's statement on the blackboard concerning object efficiency being poor. Object efficiency of many large programs has been proven (granted, with a number of iterations) to be better in space allocation when done with a compiler than have been accomplished with skin machines. When you have a 10,000 word application that uses 5 words of temporary storage, this is a lot better than a handcoder can do. Human coders just don't go through the endless process of checking what storage is available. There are many things that a compiler can do well for you that programmers don't normally do.

PATRICK: I usually leave my particular code fairly loose as far as storage allocation goes because that's the only way I can check it out. I like to leave intermediate products lying around so that I can take a snapshot of them and see what I did wrong.

DOBRUSKY: You're using techniques that are machine-oriented, which is fine.

PATRICK: If you compile tight code I defy you to check it out. If you lay these intermediate products out by giving them different names, then you can get pretty good checkout efficiency. If you tighten them up real tight, I don't think you're going to check them out any faster than I am.

CHEATHAM: I don't think this is going to be a useful argument. You can build a compiler that will lay them out during the checkout phase and squeeze them up later.

PATRICK: Now you're talking about a two-mode compiler--one that has a checkout mode and a production mode.

* : I think there's a fair amount of enthusiasm for that concept.

GORDON: There is enthusiasm for more compilers than you can write already. This tends to double the number that you're committed for.

GALLER: Not necessarily. One of the objectives of the next version of MAD is that it be done in such a way that we can hook in optimizers. One of the optimizers will be temporary storage assignment. This feature will just move in when you're ready, and every such optimizer will be optional.

TALMADGE: I think people, in general, will agree that during the period of checkout it doesn't pay to waste time trying to optimize. That is, one would like to get a checked out program that is logically correct even though it may be clumsy. Later the packing can be done by optimizers; or in a real-time application, one may even call in the best programmers on the staff and polish up the machine code.

CHEATHAM: Don't you do that already to some extent, in order to take out the debugging aids from the finished result?

GORDON: I'd like to amplify the equation you have in front of us, Bob. In your equation you have, as two separate terms, dollars to compile and dollars to assemble. These are more or less the same thing. I'd like to suggest that these be broken up into a product of two things; namely, the cost per compilation and the number of compilations per checked out program. If you are looking to reduce compile costs you can do it in one of two ways. You could, for example, design the compiler so that you could compile in 42 minutes instead of 45, or you can design it so that you can complete the checked out program in three shots on the machine instead of 8. I think that an intelligently designed source language can perhaps produce greater savings in terms of the number of compiles required, rather than in worrying about every nit-picking millisecond in each compilation run.

PATRICK: Those must be IBM times that he had in mind--42 minutes to compile, etc.

MacKENZIE: Possibly he's on the subject of how one ought to look at a compiler. It may seem ridiculous, but from the user's point of view you ought to look on it almost as a fancy loader. If the compiler performs satisfactorily as a loader, should you care whether you recompile or not? You've got to get the information into the machine. Admittedly, this may not seem attainable, but it might be something we could agree on.

GORDON: I don't think there's any one thing we can agree on.

PATRICK: And that may be the one thing we can agree on.

CLIPPINGER: You asked earlier whether anyone would be willing to make statements as to whether or not we are on the right track. I'd like to pass along the experience of one of our customers who is using FACT. This is not intended to be an indication that I think we have the right answer. It indicates merely that what we have is going to appeal to some of the users in such a way that we aren't going to be able to let it drop. One of our users started in January to write a payroll application. The application involved daily, weekly, monthly, and yearly runs. The effort represented the work of about one-and-a-half programmers from January through the end of March. The result is a complex of 16 integrated programs involving about 40,000 three-address instructions (roughly equivalent to 80 or 90,000 single-address instructions). If you extrapolated this experience, this would be roughly

equivalent to 300,000 single-address object program instructions written in a man-year. The job concerned happens to be a data processing application, of course, with a real purpose and it is in operation. That's the sort of thing the users are looking for and that's what they're going to get. It costs us manufacturers a pile of dough. Admittedly, the object code is not as good as we'd like it to be; we can improve it. When we get through improving it, it still may not be as good as we'd like, but it will be good enough so that users will use it.

There may be better ways to do it, but I think English language coding is here to stay. We've got something that is going to do the users some good.

I'm working as hard as anyone in the United States in the direction of standardization, but you all seem to be quite skeptical of the rate at which we're going to accomplish anything in the way of creating useful standards. I'm not at all certain of what the right path is toward standardization. I do know that standardizing a programming language is an extremely complex business. I happen to think it's desirable, but I don't know to what extent we're going to succeed in it.

GORDON: I question your statement that you will not be able to abandon it simply because customers like it. I can assure you that this is no necessary impediment to abandoning something. We have been through this.

CLIPPINGER: It may be a daydream that you will be able to abandon Commercial Translator.

GORDON: It may be, but it looks as though we'll be able to. There may be some unhappiness on the part of some customers.

PATRICK: Have you ever thought of taking the Commercial Translator maintenance crew and have them put bugs back into it and use that as a way to kill it?

* : They don't have to do that.

GALLER: Has IBM talked about a translator to go from Commercial Translator to COBOL to help your people change over?

GORDON: SHARE talked about that at the last SHARE meeting.

LITTLE: It's called a programmer.

GORDON: IBM has not spoken about it as far as I know.

I have a question for Dick. As I recall, COBOL will be available for the Honeywell 400, but not FACT.

CLIPPINGER: Correct.

GORDON: Both FACT and COBOL will be available for the 800?

CLIPPINGER: Correct.

GORDON: What about the 1800, Dick?

CLIPPINGER: The 1800 is logically the same as the 800.

GORDON: So FACT will be available for the 1800?

CLIPPINGER: Right.

GORDON: Good luck.

CLIPPINGER: There's no problem at all there since the two machines are logically identical.

PATRICK: We've slipped sideways into another way to solve the incompatibility problem and that is not to tamper with the machine order list. I may be wrong, but I think Honeywell was the pioneer in upwards compatibility. The 400 and 800 were upwards compatible.

CLIPPINGER: Hold on. We have never claimed that the 400 and 800 were upwards compatible.

PATRICK: Isn't the order code of the 400 a logical subset of the 800?

CLIPPINGER: No.

*: Not the least little bit.

CLIPPINGER: There is one language (EASY) which will run on either machine. The 800 and 1800 are upwards compatible, of course.

OPLER: There is tape compatibility so that certain files would be upwards compatible between the H400 and H800.

GRUENBERGER: Is the collating sequence the same?

CLIPPINGER: Yes.

PATRICK: Does Philco have upwards compatibility in the 210, 211, 212 series?

DOBRUSKY: I think that's correct.

PATRICK: And IBM is getting on this bandwagon?

GORDON: I think so, starting about 1956 to 1957.

PATRICK: When the 704 went to the 709 and 7090?

GORDON: Yes. The claim was never made in going from the 701 to the 704 and the compatibility going from the 704 to the 709 was a less than perfect solution. The 705-III and the 7080 have been pretty well compatible for some time.

PATRICK: But with a switch, isn't that so?

GORDON: The 7080 has a switch; the 705-III doesn't. The 705-III's order code is a proper superset on the 705-II. The 7070 and 7074 are highly compatible. The programming, though, is identical with the exception of things that depend on timing. Since the 7074 is much faster on the main frame, you could get into trouble if you program real tight on read and write operations. Within the main frame the logic is identical.

GRUENBERGER: Isn't it funny that all of you can bat around machine numbers so freely, but Clippinger and Phillips are the only two I have ever heard who speak so freely of the X-3 numbers.

ARMER: I was trying to do more than try to coax people to say that they thought they were doing things right. I was trying to challenge those people who say we're doing everything all wrong to indicate how things should be done differently. Barry's suggestion is a good one, but it's not really a change in what we should be doing, but an addition. That is, his emphasis on standards regarding what is a programmer.

GORDON: Yes, and stop designing languages for three-year-old programmers.

MacKENZIE: I think we ought to do two things differently, in a sense. I like the point about directing more attention at the level of users. In a way it is an argument against standardization, although I would not like to identify it as such. Lots of times you have to give people what you think they need, not what they claim they need. In the case of the language we are talking about, we are essentially talking about "stuffing" into present programmers the type of training that it took to make them reasonably good machine or assembly language programmers. I think this is a very serious shortcoming in a lot of our approaches. Another point is the constraint that exists at the hardware design level. I think a question well worth looking into is to what extent will research in problem-oriented languages reflect itself in proposed machine organizations. If you'll agree to remove these constraints, I think you'll see different types of machine organizations in the future.

GORDON: The machine organization itself, I think, is another argument against standardization, at least at this time. Take a look at committees such as the COBOL committee or the ALGOL committee. On any of these inter-company or industry-wide committees you have men who may have a darn good idea of things going on back in their plant relating to, in effect, unannounced hardware. It's darn difficult to be

able to get up in a committee meeting and say, "Look you guys, we're announcing a machine next year. But it's not going to work this way at all." You just can't get this into a standard very effectively. It's tough to say "it's this way instead of that way" and not give a reason for it. If you want any effort to go a certain way, you've got to be able to say why. If you're basing your ideas on things that will come along in the future, you can't say why. In effect, then, any of these committees will be working several years behind what any individual manufacturer could be doing. People at Honeywell can be working on systems to support future hardware that they can't talk about. It's the same with Burroughs, and even IBM.

This is a real limitation on any sort of committee action. The individual manufacturers, who can see what's coming up, can plan for it individually, but can't talk about it collectively.

BROMBERG: Are you saying that language functions may be precluded by machine design?

GORDON: Influenced, not precluded.

BROMBERG: Is it possible then that one could come up with a machine that would make things like report writers and sort generators unnecessary?

GORDON: Okay, so what?

GRUENBERGER: I don't think that was the point he was trying to make, Howie.

BROMBERG: I want to know what you're going to do to implement a language that you can't talk about because of proprietary machine design.

GORDON: I would tell you if I could talk about it.

LITTLE: There's an interesting point the other way around, though. If you once came up with a really good standard language, you would probably see a turnabout in the hardware. You may come up with general purpose languages and special purpose computers; that is, computers would be designed more and more to handle a particular language. That tends to stop the progress of machines.

MacKENZIE: It forces the progress to be directed in a particular way. For example, suppose that language A were really a standard language; certainly then some manufacturers might like to sell to the audience that was using language A. They should tend, in their development areas to produce machine organizations that could efficiently handle language A.

GALLER: Do you really think that's true? Take FORTRAN. Let's assume that history might repeat itself and that there will be a super FORTRAN. (Some of us would like to see a super FORTRAN that would be amenable to Command and Control.) Now is it really the case that machine design has been so strongly influenced by the fact that FORTRAN has been so popular? I don't know; I'm not prepared to argue for or against.

* : How much evidence is there, Bernie?

GALLER: I don't know. I think it's tremendous, though, that Burroughs took the step with the B-5000. I think it's the one really clear-cut example we have, but they may not succeed with that machine. But certainly any future machine, at least in the design stages, has to compare itself with the B-5000 to see if it can do better; or whether it should go in that direction. You hear all kinds of rumors that maybe IBM is moving in this direction, too. I don't really care whether they're moving in this direction or not, but they had better be thinking about it, and that in itself is good.

PATRICK: We've already seen new machines being selected on the basis of their compile time on a specific application and the object code execution on that same application. It's only natural that the manufacturers tend to weather vane. If you're going to do a lot of sorting, for example, maybe you ought to have a D-1000, simply because it will sort better.

CLIPPINGER: Don't start selling D-1000's. We don't make them anymore.

PATRICK: Don't they have some to spare, Dick?

GRUENBERGER: Like seven?

GORDON: It sorts like a bomb; if only you could carry the tape files over to it.

PATRICK: Can we agree that something might be done about the level of programmers and then design languages compatible with that level?

McCRACKEN: I'd like to take a certain amount of exception to that idea. It seems to me (thinking about such

matters and getting ready for some writing jobs that I have in mind) that the real problem in training a beginner in how to do good programming is not that big a function of the language. What has to be taught to a guy who has just walked in off the street? He is taught in the context or framework of some language or other but the things we spend a lot of time teaching him are not that language. We teach him such things as what is the difference between a problem and a procedure. We teach him what is the whole problem of run-to-run communications. What are control totals all about? What is data verification?

We do these things traditionally in the framework of giving him a machine language course or a COBOL course or something, but the details of that language are not the toughest thing for him to get straight on. You can rapidly verify this if you give a guy a course on the details of a language and then turn him loose on a problem. So I don't react too favorably to the idea of designing a language to be easy to train.

PATRICK: No: to teach.

CLIPPINGER: I think there are two aspects in the problem of training someone to use a language-computer combination. You do a good job in getting the guy started in understanding the language. Before we finish with him though, we have to teach him all the special things about restrictions without which he really doesn't really know how to use that language.

These have to do with the particular implementation that we have invented to put that language into operation. That part of it gets to be bigger than the part about getting a broad general feeling in the language to the point where you can readily dash off problems that work.

McCRACKEN: I understand what you said perfectly. I have had some recent experiences with this myself (in trying to work from a manual) and in a real short program I have found six important things that had to be done that weren't in the manual. What I am saying, however, is that the details of the language, plus all of this still is not the main thing that has to be taught.

PATRICK: I think you two are talking about different things. I think, Dan, you're talking about the teaching of programming per se independent of language; namely, how do you analyze a job and do it? Dick, on the other hand, is talking about a specific language embedded in an operating system in a facility with its operational procedures and techniques.

McCRACKEN: And I'm saying that teaching the details of the language is a relatively small job. Of two equally good languages the more teachable one is to be preferred, I guess.

BROMBERG: Wasn't that exactly Barry's point? He wanted to set up certain standards for comprehension of what programming is about.

McCRACKEN: Look, let's take the familiar experience that after you know one machine, it isn't hard to learn

another. It's clear then that when you were learning the first one, what you were learning was not the machine; it was concepts.

BROMBERG: I don't think we're arguing. I think the point is the same. If you're going to set up standards for programmers, just set up standards for whether or not they understand the concepts of stored programming, not whether they understand a certain language. Once they've gained a certain level of competence, then you are able to address your languages to them.

TALMADGE: When Gordon and I were having our discussions on what Commercial Translator might be, there was a certain character who kept popping up: the poor accountant who was going to use this language. We started calling him "Joe Accountant." He became a much used (and abused) character. Now, my background is scientific computing and Gordon's is commercial so that I was always faced with this character whenever a question came up as to whether or not we should include a certain item in the language. The whole point was that Joe Accountant really didn't know anything about how to use the machine, and we were designing the language for him. If we could restrict the users of the language so as to be able to demand a certain level of competence, then we could do things in an entirely different way.

McCRACKEN: And all I'm saying is that you can design a language so that Joe Accountant can use it with very little difficulty; then you have still not begun to solve the

training problem. You still have to teach him all these other things that I mentioned before. Once he has learned all those things, then he can understand a more difficult language.

GORDON: I'll go you one better. If you can design a language that Joe Accountant can learn easily, then you're still going to have problems because you're probably going to have a lousy language.

McCRACKEN: That's opinion.

GALLER: Not necessarily.

GORDON: Maybe you've been living in an ivory tower, Bernie. I've been working with Joe Accountant for a long time. Incidentally, I want to apologize to Dick Talmadge for last year.

GALLER: All I'm saying is that it's not necessarily a bad language to work with just because you can teach it.

PATRICK: It may be rather weak though.

GALLER: Again, I say, not necessarily. I look at MAD. It's easy to teach, and it's wonderful to use.

* : Whom are you teaching it to though?

GORDON: You're not teaching it to Joe Accountant.

PATRICK: He couldn't get into the University of Michigan.

GORDON: You're teaching it to Joe College on campus.

GALLER: All I said was, here was a counter-example.

McCRACKEN: What you're teaching to Joe Accountant is not a language in any case. Further, ALGOL is easy to teach.

GORDON: ALGOL is not easy to teach.

MacKENZIE: Bernie can go off and teach things to students in college and have them understand them. They can be fairly complex things. You can go to a group of "professional" programmers and try to teach them these same things and run up against a brick wall. It has to do, I believe, with the open-mindedness with which people approach the subject. We've been criticized a great deal for wanting to teach people the syntax of a language, as opposed to conventional ways in which they might want to view the language. By and large, there has been a great deal of success with this method at the university levels. Many of these people did not have prior programming experience and after a few months came up with a much better understanding of how the language "worked" than the people who wanted to approach it by some method akin to the way these things are normally done.

GRUENBERGER: The earlier you catch them the better it is.

MacKENZIE: Sometimes you shouldn't give them what they think they want; you have to give them what they need and you may have to be pretty arbitrary if you're convinced you're right.

GORDON: I still say that when you're working at the university level, particularly with math majors and engineers, you're dealing with a pretty select group. When you get out among the great masses who are going to be programming in

SPS, these guys...

MacKENZIE: I think you're missing the point, Barry. I may be very presumptuous, but I think you can approach these people on the basis of "this is what you need to know to be able to use this thing" and they either never know the difference or they are willing to accept the approach on faith. When you go to a user or any environment of people who have done previous programming, frequently you find that they have preconceived notions--preconditioned ideas of what the terms are under which they will accept any new thing.

LITTLE: I think Barry may have a point there. I don't really know; I'm asking those people who have university experience. Of all the people you teach to use the machine, what percentage breakdown do you get out of particular fields? How many sociologists do you have programming a computer, or economists, or psychologists?

GORDON: Never mind those guys. Let's take the guy who graduated from high school at the age of 20 by going to school at night. Perhaps he left high school at the age of 14. He has never had elementary algebra and doesn't know what a negative number is. But after 15 years of running a tabulator he is suddenly one of your programmers.

LITTLE: I agree with you there, but I think by the nature of areas that people are in, sometimes, they are more open to learning certain things than other people are.

Take open shop people for example. We have an open shop

operation here at RAND. I would suspect that you get a lot more reasonable response out of the engineers in the corporation than out of the psychologists, for example, but in a real world we have to go out and do jobs for these people. If you're going to design a language or a system that is to be used by these people, please recognize that there is a broad spectrum among any group. As a matter of fact, some of the really big jobs are done not for people who could grasp it if you gave them the chance, but for people who may never understand it. Furthermore, they don't have much of an interest.

PATRICK: The military is the best example you have there. They couldn't care less what your problems are. "Just tell me what supplies we've got out in the warehouse." They don't want to know how you do it.

OPLER: We consider that the COBOL user is a skilled systems analyst who understands both the machine and the system. There is the difference between the training level of a person who will use a data processing language well and the training level required to use an algebraic language.

Instead of visualizing the bank clerk knowing the complexity involved in the data description and the interplay between it and the machine procedure, we feel that probably the best user of the language will be the man who has not avoided systems analysis in the past. He can use such languages to get on the air quickly and to produce 250,000 instructions per year.

McCRACKEN: How long does it take to teach him systems analysis compared to how long it would take to teach him COBOL? Much longer. The language isn't the problem.

CLIPPINGER: What you're saying is the manufacturer ought to tell the customers that it takes good people to use this well. If you want to get by with very poor people, you do it at your own risk. You might expect to get something done, but it could be pretty bad.

GALLER: I was talking to some people who were getting ready for the 1401. They had been sending their tab people to school to learn how to program for the 1401. They were astounded when I recommended that they get at least one person who knows programming and doesn't know their business.

GORDON: But the salesman said you didn't have to!

BROMBERG: National Cash Register put out an interesting document that they called the "NEAT COBOL Manual." It's been a sort of vogue the last 12 months or so for everyone to flood the market with their COBOL manual, but NCR deviated slightly for about one-quarter of the manual. Instead of talking about how well they had implemented COBOL, they talked about systems design. They made the following comment, which I thought was terrific. They said, "COBOL is not a substitute for good systems analysis." This is just what I think the sense of this group is.

McCRACKEN: There's one thing that I think that COBOL is efficient at and that's wasting one hell of a lot of machine time if you use it wrong.

LITTLE: One thing we seem to be agreed on is that the training problem is a very important one. It seems to me that there's a lot more work going on in building compilers than there is in the problem of training. No one seems to be much worried about how you make up good training courses and make up good material for them.

ARMER: It's even hard to get people interested in this area.

GORDON: I'd like to remind everyone about the fable about the emperor's new clothes. You remember the con man convinces the emperor it's a great suit, that only the pure in heart can see it. So the emperor winds up walking around naked and no one has the guts to say so. And finally some kid says, "Hey, look! He's naked!" and everyone realizes they've been had.

To put it bluntly, I think it's about time somebody had the guts to get up and say that the emperor is naked.

*: Who is the emperor in this case?

GORDON: I think the emperor is a very large segment of the computing industry. That includes users.

PATRICK: Like the guys that believe these ads.

BROMBERG: I don't understand why you speak so disparagingly of users. I would like to go on record as saying that we at RCA like users.

GORDON: I'm not sure you can do anything about the advertising boys or the salesman, but in spite of what they

do we have a responsibility to see what we can do to bring some order out of the chaos to produce useful equipment to do jobs and to produce useful programming tools. We must get the word out as best we can. I realize this is very idealistic.

LITTLE: It would seem to me that if anyone is real eager for a good set of training devices, etc., it would be the Department of Defense and the military. Nevertheless, we still train people by the old buddy system, although SDC has more formal training. Work along these lines would seem to be very valuable.

PATRICK: We have some facts, gathered at RAND (see Table 2). We were concerned as to just how much we had to pay for higher level languages. We seem to think we know what we're getting in higher level languages. (Although we don't really know.) We use FORTRAN here at RAND and we use SOS. It's interesting to look at the time fractions and how they're distributed. Before Armer's crew went out to get these numbers we wouldn't have guessed they'd be anything like this.

This is a summary sheet. The first pair of rows across the top are the number of jobs that took between zero and five minutes. The second pair is the proportion that took between zero and ten minutes (which include, of course, the first set).

This is a 4-month summary on RAND's 7090. Total machine time is 565 hours. Total jobs number 9561. Of these 9500

RAND CORPORATION - COMPUTER SCIENCES DEPARTMENT

Computer Operation Study, November 1961 through February 1962

Time Increment (Min)		No. Jobs	Time(Hrs.)	% Total Jobs	% Total Mach. Time	% C	% P	% Non-Exec.
00-05	FTN	2922	62.64	50.6	18.2	74	26	48.8
	SΦS	1884	54.66	32.7	15.8	92	8	37.7
00-10	FTN	3126	85.93	54.2	24.9	74	26	42.6
	SΦS	2186	88.03	37.9	25.5	91	9	32.4

Note: Total Jobs (Sample) = 5768
Total Jobs (4 Mon.) = 9561
Total Machine Time (Sample) = 345.34 Hrs.
Total Machine Time (4 Mon.) = 565.4 Hrs.

Note: "C" = good code check
"P" = good production (Some jobs are labelled "C" and should be "P.")

Note: The sample size does not include open shop, "outside" customers (i.e. Non-RAND), and jobs that reported no breakdown as to system usage.

Table 2

jobs, we had complete data on 5700. The time for them was 345 hours--so it's a fairly good sample.

Of the jobs that took between zero and five minutes, there were 2922 done in FORTRAN and 1800 done in SOS; the corresponding times were as shown in Table 2.

"C" means good code check; this means you made a run on the machine and compiled (or assembled) and tried to execute--something useful to the programmer in developing a new running code. "P" (for Production) means there were no programming changes from run to run.

The last column shows the per cent of time not spent in executing. The 48.8%, for example, is a percentage of the 32.64 hours not spent in executing. It includes loading and dumping and waiting for tapes to rewind, and so forth; that is, getting ready to execute.

ARMERDING: That includes, also, assembly time and compilation time.

BROMBERG: Recompilations and reassemblies--are they in % Non-Exec, too?

ARMERDING: Everything that isn't in the execution of the program.

BROMBERG: You can execute the program a number of times (in between each compilation you execute). That's execute time (even though it's not execute for production)?

ARMERDING: We read the clock when execution starts and when it stops--this is the clock time other than that.

LITTLE: On FORTRAN compile-and-go, the execute time would be considered execute.

BROMBERG: Suppose you get the wrong answer?

GORDON: I don't understand how you can get 74% checkout and only 49% non-execute. Is execution during checkout still called execution?

ARMERDING: Yes. Let's look at the percentages under the second and third last columns labelled per cent C and per cent P. 74% is code-check and 26% is production for FORTRAN jobs running less than 5 minutes. In SOS jobs 92% is code-check and only 8% production. When we go to jobs that run as long as 10 minutes the percentages change hardly at all.

LITTLE: Do we have one big SOS job done yet? Maybe that's our problem.

CLIPPINGER: This means you check your FORTRAN programs three times and then you run them once.

PATRICK: Remember these figures are just for one shop and a fairly unique one at that. But this says we spend an awful lot of time in preparing codes and not very much time in running them.

McCRACKEN: It also says you check out your FORTRAN programs faster.

ARMERDING: No, that doesn't follow.

GORDON: It sure looks like FORTRAN helps improve your checkout.

LITTLE: That may be true but you have to also realize that FORTRAN jobs are basically smaller.

McCRACKEN: Checkout is three to one on FORTRAN and nearly 11 to 1 on SOS.

ARMER: No, it might be misleading. The SOS jobs might take longer than 10 minutes as a rule. If most of them took longer than 10 minutes then on the basis of these statistics that ratio would be infinite.

BROMBERG: Was the purpose of this document to say something good or bad about common languages?

PATRICK: Neither. I think you can conclude just one fact; that if you design a hardware-software system and expect to run a lot of production on it, that principle is not very well borne out on the basis of this installation's figures. This installation seems to run a lot of code-checks.

GALLER: At the university we probably have more code-checking than any other place around. Quite frequently when our students get a program running that's the end of it because that was the problem.

GRUENBERGER: Yes, that's the way we operate, too.

GALLER: I looked at our log for January and brought some figures along that were very surprising to me.

In our January billing we had 6064 runs. This is on the 709. I don't have the breakdown of these jobs in terms of MAD, FORTRAN, and so forth but I do have these figures for 242 hours of use. Execution took 167 hours or 69% (in March

that was 78%); MAD translation time took 46 hours or 19% of the time; FORTRAN took 20 hours or 8% of the time; our assembly program took 9 hours or 4% of the time. The average job length was 2 1/2 minutes (although it seems that every time I bring a visitor in to see the machine there's a 45-minute job on at that time).

McCRACKEN: Doesn't MAD compile faster than FORTRAN?

GALLER: It's possible to have a shop where the translation is very fast, leaving plenty of time for execution.

McCRACKEN: Why can't FORTRAN compile faster; that's what I've been trying to find out all this time.

GALLER: You have to look at these figures and see that the time we're spending in compiling and so forth is really a function of the particular translator.

PATRICK: That's correct.

GRUENBERGER: Are you getting a 7090?

GALLER: Yes, we're getting one in August.

OPLER: Bernie, do you have any problems that have been compiled by FORTRAN that run longer than an hour?

GALLER: I would think so.

OPLER: Do you compile these by MAD too?

GALLER: I doubt if we've done the same problems both ways.

OPLER: The point I'm getting at is this. While FORTRAN has obviously been sub-optimized for those people who are doing many, many compilations on small jobs, it is much more

directed toward long production jobs, where they are concerned with, for example, the time required to go through a set of partial differential equations or a long linear programming problem.

GALLER: We have plenty of people doing partial differential equations in MAD.

OPLER: MAD may have been maligned. I understood that FORTRAN makes tighter loops on...

GALLER: Definitely. Most of our programs will run anywhere from one to two times as slow. On a real big program, that would make quite a difference. But that's only if you are able to run the same problem many times between compilations.

ARMER: Do you have a MAD to FORTRAN translator?

GALLER: It's too hard to write.

* : Could you write it?

GALLER: It would be very difficult.

McCRACKEN: That's what Bob Bemer said about FORTRAN and ALGOL, too.

GALLER: It's easy from FORTRAN to ALGOL but hard the other way.

McCRACKEN: Oh sure.

PATRICK: You can't put a 2-yard load of dirt in a 1-yard truck. We didn't mean to bring out these figures to kill further discussion. We thought it would stimulate further discussion.

C O F F E E B R E A K

ARMERDING: I'd like to make a rash statement. In my experience the process of problem solution with a computer/COBOL combination is unteachable, at least to the people we think we've been teaching it to.

LITTLE: Are you saying COBOL is not teachable?

ARMERDING: No, I'm saying solving problems using a computer-magic language combination is not teachable to the great unwashed masses.

McCRACKEN: What do you mean by saying not teachable?

ARMERDING: That I can't pick up the man off the street or the 407 operator who is about to become a programmer for the 1401 and make this transition.

* : Ever?

McCRACKEN: Unless he's intelligent. You mean you can't teach it to him by giving him a set of lectures.

ARMERDING: Bernie can take his people at Michigan and teach them MAD and how to solve problems using MAD. They go away fine and they come back and they're able to solve problems. We can't do this. I'm basing the statement on the experience that I've had. Your work notwithstanding, Dan. And I've done a good deal of this teaching recently using your FORTRAN text. I've done my darndest and I don't think anyone else is going to have any better success.

GALLER: But if we can't teach it we're doomed. We've got to teach it to them one way or another.

* : But we don't have to teach it to everyone.

ARMERDING: We've already decided that the 44 people that Dan's friend employed don't really represent the solution. According to Cheatham, who has had some experience in this sort of thing, and according to others I have heard, what he'd rather have is three top-notch people. Presumably he could hire them if he'd pool the salaries of the 44 and split it three ways. He would turn out the same work and get a better compiler and the whole job is done better all the way round.

GORDON: Are you talking about the use of these or their construction? You started out by talking about training users.

ARMERDING: All right, now I'm talking about the construction of the compiler, but it's the same principle. If you talk to a guy like Jack Little who has to implement real live problems on the machine every day, he too would rather have a few top-notch people that a whole stable full of no-goods. So my question is should we even try bringing in these great herds of people and training them?

McCRACKEN: It's only by starting with big herds that you find out who the best ones are. You've got to give a lot of people the first course in order to find out who should go into the second course.

GORDON: That's not true. There's a tremendous difference in level of competence. I don't intend a plug, but you could give them the IBM programming aptitude test, as we have done for years, and you can eliminate most of the tab operators

and file clerks and nephews of people who have worked in the company for 20 years. You bring in guys with certain basic minimum requirements, like literacy and the ability to count up to 2-digit numbers.

McCRACKEN: And having done that, of what's left 99% of them will turn into the kind of guys who can do COBOL in three man-years.

GORDON: But they're the kind of guys who can at least program applications in a reasonable programming language. Even so, you ought to eliminate some of them. The point is, we're not even trying to do that. We're trying to teach the guys who have been wiring control panels or pushing buttons on punched card machines for years.

ARMERDING: Who is?

BROMBERG: We're not trying to teach them to write these things.

GORDON: No, to use them. You cannot teach 500,000 guys who are basically machine operators to become competent programmers through the use of any magic language. Isn't that what you were saying, George?

ARMERDING: That's what I was saying.

BROMBERG: Neither can you teach them to be competent programmers through the use of any unmagical language. However what you can get from the use of these magic languages is some few number out of this great hoard who can carry the ball and keep enough of their end up to bring the rest along with them. I just don't understand why you say that these

languages are so difficult to be taught.

ARMERDING: I'm not saying that. I'm simply calling attention to whom we're trying to teach them to. I'm simply saying that building magic languages is not going to help our education problem.

McCRACKEN: Oh, I agree with that.

ARMER: He's simply saying, "Don't expect magical languages to solve your training problem." In other words, don't believe those ads which say that programmers are no longer necessary for getting the job done.

LITTLE: And if you once assume this, doesn't it also say something about how you go about constructing these languages?

GORDON: That's a good corollary. If you give up the idea that these magic languages are going to get production out of your three-year-old programmer then you can also scrap the idea that the languages should be developed for a three-year-old programmer.

McCRACKEN: Now I think we have said something.

PATRICK: I'd like to raise another particularly obnoxious point along this line. It doesn't seem to me that the magic languages help my documentation significantly.

KOORY: Amen, brother!

LITTLE: I'll speak for the only other implementer around here and also say amen.

PATRICK: You've still got to describe the job, and usually this description comes in something like flow charts and some standard symbols. This can all be built into the

compiler code if you wish but that makes it compile pretty slow; it's kind of bulky then.

McCRACKEN: Whose documentation problems? Are you an implementer now?

PATRICK: No, I'm talking about applications.

McCRACKEN: Gee, there are other users who don't say that. They say very carefully that the magic languages don't solve the documentation problem; you still have to get a bull whip on the programmers. But having gotten that bull whip working then it is a bit easier in say, COBOL, than it is in, say, Autocoder. Not only that, the program is the documentation.

PATRICK: That's just the point. The program is <u>not</u> the documentation.

GORDON: The program is the bull whip. What you say to them is, "If you don't get those flow charts up to date, we'll make you write in COBOL."

LITTLE: You can put comments on your coding sheets all you like but I don't think that's a substitute for what I call a minimum set of documentation. I still want to see flow charts, a narrative of the problem, symbol definition sheets, and so on.

GRUENBERGER: And test cases.

McCRACKEN: On the other hand there is more than one user who uses flow charts written in COBOL as the documentation and the program. They keypunch from that.

PATRICK: I heard an interesting thing about that down at STL. The people at STL gave an interesting pitch the other

day about how they're using flow charts written in COBOL and how they keypunch directly from them. The fellow said that it sounded real good but in walking through the key-punch room he didn't see any of these huge flow chart sheets lying around on the 026's. What was going on, it turned out, was that some girl was copying from the flow chart sheets onto key punch sheets.

McCRACKEN: It's a delightful story but it's not universally true. There are places that work from the original sheets.

DOBRUSKY: What rating do they give their keypunch people? Are they called programmers?

McCRACKEN: No, they have a flow charting convention.

BROMBERG: The interesting thing about this is that now the proper perspective has been reached. It is a clerical function at best, to go from the problem definition sheets through to key punching.

PATRICK: That may be. There have been some of us who have been saying all along that there should be the equivalent of engineering aides following the good programmers around. I can do about 8 times as much work if I have four more pairs of hands. I don't have to keypunch my own work. But don't tell me I don't have to document. The same amount of work is being done, but just by different hands.

CLIPPINGER: If you go talk to Maurice Halstead, he'll hand you a thing which is a printout of the input to his

compiler, as a definition of his compiler. I don't think it's very pleasant reading to find out how his compiler works, but I claim it does tell you what his compiler does. You can take it and use if it you want to. It's a lot better than working from machine language.

MacKENZIE: In our case, I don't know how well we'll solve our documentation problem but I am sure we'll get a much better solution by the means we are using, namely, by writing the processors in a machine-independent language, than we would achieve through any of the earlier techniques.

BROMBERG: Are you going to do this with your COBOL compiler?

MacKENZIE: Yes, we're going to write COBOL in B-5000 Extended ALGOL.

LITTLE: I put this whole argument in a class with putting comments on SAP sheets. I can't argue that it is not helpful and if carried through, very helpful, but it is not a substitute for good documentation, the kind that you need for people to understand, modify, and pick up that code.

MacKENZIE: I think you touched on a real problem though. No one says that this is the only way to do it or even the best way to do it. But by just letting nature take its course, you get a much higher level of documentation, this way, than with other methods.

McCRACKEN: It may not be perfect, but it's better than what we've got.

BROMBERG: It's a great check-out technique.

GALLER: One area that's not covered in this matter of the program looking like its own documentation is that it's still very local. The global statement is the description of the relationship between the parts; this has to be done separately.

McCRACKEN: All right, at least you've got good documentation at the detailed level. You don't even have that now in Autocoder.

LITTLE: Think back to when you used to do this. In the first pass through, I, like many people, write pretty good comments; then I find my first mistake. The line has a big long comment on it but it usually isn't to be found on the correction card.

McCRACKEN: That's not what I'm talking about.

LITTLE: But if you're going to use a bull whip, let's get what's necessary.

GORDON: Jack, you're arguing against yourself. If you had been working in COBOL, the first time around you would have named this thing with some long mnemonic. That means that if you change that card when you find your bug you must write down the same name.

LITTLE: I have not done this in COBOL. I've done it in machine language and in FORTRAN.

MacKENZIE: I think you're thinking more of the things you can write down as comments. I think you're missing the

point that there are a lot of things that you no longer need to write down as comments because they appear in the basic code.

GORDON: It would be better to say that they get buried in the code.

PATRICK: If you do it the way you're talking now then I will have long names and I will define each of them separately. If I want a fairly compact code...

McCRACKEN: Source code.

PATRICK: No, I mean object code. I will use names like working cell 1, working cell 2,...

McCRACKEN: What do names have to do with the object code?

PATRICK: I'll give each intermediate product a different object name.

* : That has no effect on the object code.

PATRICK: It will assign them a different cell if you have a stupid translator.

* : No, it's the data organization which does this job.

PATRICK: I seem to be having a hard time communicating.

* : That's for sure.

GORDON: We need a good standard language.

PATRICK: What we need first is a glossary. What I was saying is that if each cell has a separate distinct, unambiguous name, then the thing might be fairly easy to read and could be self-documented. This means that every name gets a cell or set of cells assigned to it in the object code.

GORDON: This is typical of business applications where you're dealing with records and arrays and eventually all of them...

McCRACKEN: That's just opinion.

GORDON: Look, programmers get stuck with things like this. A file clerk has made up a data description of a file and a programmer is presented with it and is stuck with it. He has to use the name that is assigned to the data every time he refers to it. In effect, this is a bull whip. If you turn the systems analyst loose, he's going to assign names to the files and the programmer is then stuck with those names.

BROMBERG: You can change the names.

GORDON: How do you do it?

BROMBERG: How does your system handle the "REDEFINE" clause?

GORDON: You'd have to re-write the whole data description, characteristics and all.

LITTLE: And if you use it you're back to a symbol definition sheet, which is just the way we always did it before.

PATRICK: If I defined four separate fields, then the source statements as I write them are relatively self-documented. If I define two general purpose fields, then the source statements are not self-documented. In one case the object code uses less temporary storage cells than the other.

McCRACKEN: I'd summarize my statement of the case by saying that getting good documentation takes a bull whip in either case. I think that COBOL makes it easier for the manager to apply the bull whip because the documentation is, at least to a certain extent, built into the procedure division by the very act of writing it.

PATRICK: I still think that although it may help me some in the documentation, that it's going to cost me in the compile time.

McCRACKEN: I don't see it.

* : I can't tell from your example that it has anything to do with whether it costs you in compile time. In most of these languages you can name the same field by as many names as you want to.

GORDON: Just in Commercial Translator, not in COBOL.

GRUENBERGER: Doesn't it all come back to this same subject of training? COBOL, in the hands of a master, is a beautiful tool--a very powerful tool. COBOL, as it's going to be handled by a low grade clerk somewhere, will be a miserable mess. It's going to take 20 times as long to compile and 300 times as long to execute because he's going to manage to ruin it. The guys you are writing to, Dan, are just not as smart as you are. They can distort anything. This is true at any language level. We've surely seen it back at machine language level, we've seen it in FORTRAN; there is no reason to believe we won't see it with every one of the magic languages.

GALLER: Could I ask about this unanimous thing that went by before that I didn't get a chance to vote on because I didn't understand it? Did I hear you say that you thought that half the people were at such a low level that it doesn't pay to write the language so they can read it, so you write it for the few who can? What are we supposed to do for the other half? I just came from a city where I gave a lecture and the people I was addressing were supposedly the cream of the city. It was a big city, too. And they told me that they couldn't read the Communications; it was too hard for them. These were people who were programming for 1401's and so forth.

McCRACKEN: Of course; I can't read the Communications either, for that matter.

GALLER: But these fellows told me they don't read any of the literature in the field because they can't read it.

PATRICK: If they tried reading the Information Algebra article from the current issue, I'll have to agree. I think I probably could have read that article, but why bother?

GORDON: The guys Galler mentioned were talking about the personnel notices.

GRUENBERGER: Maybe I could answer this by something that George referred to a little while ago. In the classes I teach I am continuously rocked back on my heels by the things that trouble the class; things like a three-digit number multiplied by a two-digit number is going to produce

a five-digit number. I have to stop my class cold and take about ten minutes out to explain such things. I would have thought that they learned things like that perhaps in the fourth grade. Then again, I might assume that they would know that you could multiply a number like 19 x 21 by squaring a number in the middle and subtracting something and everyone looks blank. Maybe I'm just critical of the whole educational system, but it seems to be these little things that form the real stumbling block.

George's point was that in dealing with the great unwashed masses we've been talking about, the magic language does not help. They don't solve any of the training problem and I think they obscure it tremendously.

ARMER: Particularly when the manufacturers of this equipment tell the buyers that they will solve all these problems.

GRUENBERGER: Like on the board over there in that ad.

GALLER: But the corollary seems to be that then we should upgrade the languages. O.K. But that still leaves the question of what are we supposed to do with the lower half of the mass of people.

PATRICK: That is probably the subject of another symposium. It's also the thing that should be the subject of another major effort. I don't think you should try to solve this kind of problem with a programming language.

GORDON: Like social responsibility of computing people.

LITTLE: Right now I think the programmer is being asked to carry a lot of people on his back. Not only does he have to get the job done but he's asked to use a language that is designed for an ape. Added to that the government has bought different machines and put them back to back. How many of these things can you overcome and still do a reasonably good job? Especially if you perpetuate them. You continuously expect the programmer to bear more and more of this burden. Programmers can't keep making up for stupidity all the way along the line.

PATRICK: I have a fact.

CLIPPINGER: Make sure you do. Last time you said you had a fact, you didn't come up with one.

PATRICK: All right, this is a real fact. Speaking of irrational actions, there is a 90-column card reader available for the 1401. That to me is an irrational act and along the lines of what Jack called perpetuating stupidity.

OPLER: Moreover, you don't even need one. It's possible to write a program that will allow you to read a pattern of 45-column holes into an 80-column reader and decipher it.

ARMER: We did it here.

OPLER: Sounds like they're wasting a lot of money on hardware.

GRUENBERGER: Yes, but it has been done.

LITTLE: I'd like to add a comment about teaching. Both George Armerding and I teach at Santa Monica City College at

night. Some of the problems the students have I attribute to a lack of motivation. They are interested but their job doesn't depend on it. I think a tab operator who, if he doesn't make it as a programmer, might get fired, is in better shape to learn than some of the students I get. Is it different when you're teaching people who have to learn, whose job depends on learning?

McCRACKEN: Sure, they still have a lot of trouble but they work harder.

* : It all boils down to the fact that you can't have a language which is all things to all men. Maybe one answer would be to have these languages at all levels. If you're going to have to teach Joe Tab Operator, maybe what you need is a Joe Tab Operator type language. Let him use it for his particular application but why saddle us brilliant programmers with it?

GRUENBERGER: But they aren't peddled that way. Each magic language claims to be all things to all men and that's one of the things that makes me a little red in the face.

McCRACKEN: I think one of the problems is that we're trying to live down some totally irrational claims for these languages that never were true.

OPLER: I think we've run this subject into the ground. I'd like to open up a new one.

The question I have is this: if common languages are really going to be the thing, will they be able in time to

be able to encompass more than the nice job in the middle?

McCRACKEN: And if it never could it would not constitute a total indictment of the languages.

OPLER: We might put the question another way. What percentage of the total range of problems will magic languages eventually be able to handle?

LITTLE: We see evidence today that people are trying to push these magic languages into such areas. Take Command and Control as an example. Even leaving out things like training, would you use one of the current magic languages in this area?

GRUENBERGER: If a given language has no intrinsic goodies and is automatically lousy then we shouldn't even consider it. But if it has all the goodies in the world we're still supposed to be considering the question of whether or not it is good to have it common. I suspect that a lot of our discussion has been dealing with intrinsic goodies rather than with commonness.

PATRICK: Koory's recent experience indicates that there is some hope of pushing these higher level languages into the area of gigantic applications.

CLIPPINGER: That's <u>you</u> saying it. Does Jerry say it too?

PATRICK: All I said was that there is some hope.

KOORY: I'll agree that there's some hope.

OPLER: We've done a job of approximately the same

scope using a higher level language that is not a common language. We produced something on the order of 145,000 machine instructions in a relatively short time. All this really proves is that we can say, "Hurray for higher level languages."

* : It has nothing to do with commonness.

PATRICK: I'd say that the last three topics we have discussed have been categorized by, "Hurray for higher level languages," and have nothing to do with commonness. (Documentation, training, and scope of the problem.)

CLIPPINGER: But about three-quarters of you wrote off the documentation. I don't see how you can draw any conclusions from this discussion.

PATRICK: All I'm saying is that if our words had any meaning it was with respect to higher level languages, not common languages.

DOBRUSKY: I think we did reach some agreement that higher level languages buy us something. Perhaps we can't pinpoint it with names but the utilization and the training and what not indicates that it buys us something. Now if we continue to proliferate these languages for every application we will never know what are the attributes needed for Command and Control in a language.

CLIPPINGER: You have a better chance of knowing if you proliferate them than if you don't.

LITTLE: Isn't Bill saying that you've got to be able to analyze what it is they are doing for you?

DOBRUSKY: That's right. You have to be able to establish measures on what it is that is required. If, in choosing a common language, you find complete inadequacies in it, fine. We will eventually know whether we need an extension of the language we are trying or whether we need a new language. Right now I think each application has such a large subset of the other existing languages that we are just tilting at windmills.

GORDON: I'd like to take one last swipe at commonness. There are, I think, three levels of commonness; three dimensions, if you will. You could think of a programming language that is going to be common to both a 1620 and a 7094. Clearly, such a language is either going to place a tremendous burden on the 1620 or sell the 7094 tremendously short. Secondly, you could think of a language which is suitable both for matrix inversion and Command and Control work as well as payroll and inventory problems. Clearly, such a language is going to be pretty poor for one or all of these applications. Thirdly, you can think of a language that is going to be shared by your top level programmers as well as by your retread tab operators. Clearly, this language is either going to be hopelessly binding on the better programmers or it is going to swamp the retread tab operator terribly.

If you go further and try to think of one language that will cover both extremes in all three of the dimensions I named, you'll have a pretty hopeless task. To the extent to which you try to satisfy these mutually conflicting aims in

these three different dimensions, you will weaken the language, and cut into its utility for any given application or use.

McCRACKEN: I think I agree with your first point, I'm not sure about the second, and I know I disagree with the third. I am thinking specifically of ALGOL. In ALGOL it is perfectly possible to take a subset of the whole language and teach it to people with a great deal less trouble than you would teach the same amount of computing power in FORTRAN. Then if you want to go on and use the rest of the language it can do things that we don't even know how to use yet. It's good enough for anyone's purposes. I think that this is an extremely desirable feature of a language.

GORDON: You're dealing with at least two different languages though, Dan.

McCRACKEN: No, I don't think so. You could take my ALGOL book and you could read three chapters or five or all eight.

GORDON: But the assignment statement is not the ALGOL language. You're really dealing with two languages, one of which is a subset of the other. You may or may not get away with it.

McCRACKEN: It's the same compiler though.

GORDON: For that matter, we've got FORTRAN, Commercial Translator, COBOL, and what-have-you all in the same processor. That doesn't make them the same language. Such a thing on the 705 even allows you to intermix. I'm not saying you should, I'm saying that you can. It's even rather cheap.

It doesn't take more than an hour for any one compilation; you can actually mix your statements and have one statement in COBOL and another in FORTRAN, and so on. But they're different languages, they just happen to be intermixed in one processor.

PATRICK: That sure sounds like a dog.

GALLER: But you can't go from one machine to another with such a language.

GORDON: Well it turns out you can. Some of my best friends have gone from 705's to Honeywell 800's. Of course, there's one thing required to do this. You have to have programmers.

GALLER: No, I meant you can't take your programs from one machine to another.

GRUENBERGER: He means it isn't common in the first sense you mentioned.

GORDON: That's right. If you stick to one machine family, like 709/7090/7094, you can think of common programming systems across such similar machine lines without losing much effectiveness.

GRUENBERGER: It's the same as the SAP story that Bernie started out with this morning.

GORDON: Right. If you take a small enough universe, you can establish standards across it without it costing you too much. The wider the universe gets the more it is going to cost you to standardize across it. This is true of machines, of applications, and of programmer competence.

GRUENBERGER: Howie, we could just as easily have used 301/501/601.

BROMBERG: I appreciate that. But the point I'd like to make is that as long as you do go upwards you don't have to go horizontally across. You made the point about machine sizes. This can be accommodated very easily...

GORDON: It's not just machine size, it's machine characteristics, like binary vs. decimal, character addressable vs. fixed word length, and so on. When you go from the 1620 to the 7090 you have a basically different philosophy in machine organization.

BROMBERG: Perhaps we could categorize by size and application...

PATRICK: No, I don't think so. The guys that are working on the 1401 can have just as difficult a problem as the fellows working on the 7080. They've got great big jobs and many, many reels of tape which have passed over the heads over and over again. The only reason you hack up the job is that the machine is small. It's tougher to program for a little machine. You can't say that he doesn't need it because he only has a 1401. The only reason he has a 1401 is that he can't afford anything bigger.

BROMBERG: I'm just wondering why all these things are points against the notion of commonness, as far as languages are concerned?

GRUENBERGER: Because Gordon named six fields that are mutually incompatible. If you foster one, the other five

have to yield. If you make it work for the man who has a small scientific type problem on a character addressable machine, then it won't be efficient on a different, perhaps larger, machine.

BROMBERG: And I'm saying that we could look into this problem. Facts are facts. And there is a practical world. FORTRAN exists and a lot of people are using it. There will be a thing called COBOL and there will be thousands of applications. What are we doing at this very moment to assure ourselves that the use of these languages is going to be proper and effective, and the languages will grow, that they will be responsive to change, that they will not have too many inefficiencies due to their commonness, that progress will not be frozen--is not effective maintenance the solution to all these problems?

We have talked about the situation before a language exists and we say, "What do we need for a Command and Control language?" Then we say, "We already have this particular language and it may be no good because it doesn't have some of the goodies that JOVIAL has." Let's consider that we have two languages now, and they are widely used and very popular. What are we going to do with them? They exist; we can't just say, "I'm against commonness." The languages are here.

PATRICK: It sounds like we need a rational program to get from A to B, where we're now at A. But the rational program doesn't seem to be to standardize at the stroke of a pen.

GRUENBERGER: Another thing we seem to need is some really elementary research. It's amazing how few honest facts were brought out today about common languages. For example, we don't seem to know any facts at all about the efficiencies of these languages in any sense that you want to describe efficiency. We haven't compared figures between MAD, ALGOL, NELIAC, and what-have-you. We've simply never measured these things. We've done no statistical studies either of machines, programs, or people. We might well consider exploring this large area of research before we jump.

BROMBERG: I think one of the problems is that there exists no convenient mechanism wherein all of us in this business can have a voice that is heard by the standards people and by the language extenders. For example, I think it would be very nice if we had in the FORTRAN or COBOL maintenance group a huge enough corresponding world whereby those of us who are engaged in other forms of languages could interject our ideas into that which is so very popular. I don't think we should disregard the popularity and go on our merry way creating more and more languages unless it is an attempt to coalesce as much as we can.

OPLER: Howard, I wish you could be present at a meeting of the SHARE COBOL group. These meetings are usually held in an auditorium that seats about 500 people. There are hoards of people there and they're all getting up and making suggestions like, "Why don't we take this statement out of the language," or, "Why don't we put this statement in the language," and so on.

Now this language is fairly sensibly held and maintained. But you can get too closely coupled to your own feedbacks. There are 100 suggestions proposed at every meeting. It might be better to have a small knowledgeable group that pays attention to the current needs rather than flinging open the doors and having everyone making suggestions at once.

GORDON: Howard could attend SHARE meetings quite easily if he wished.

PATRICK: Howard made a plea for a rational approach to this and you have turned it around and cited one irrational approach, Ascher. The open forum is not a very rational approach.

BROMBERG: What I'm really looking for is some practical form of language maintenance. I think it encompasses some of what you mentioned; namely, having a small nucleus who are actually doing this work.

McCRACKEN: You keep coming back and saying that COBOL maintenance is no good. I thought it was quite good. Is this not true?

* : I think COBOL maintenance is quite unsatisfactory.

PATRICK: It seems to be very slow to respond, Dan.

BROMBERG: Again, smallness by itself, is not a sufficient prerequisite for getting the job done. You have to have desire, knowledge, experience, and time. I would maintain that none of these exist on the COBOL maintenance committee in sufficient quantity.

PATRICK: Maybe we're ready to drag out the old proposals for a data processing institute. This is the sort of thing where you could get hold of these guys and let them do it as a full time assignment.

McCRACKEN: You mean an effective ACM.

PATRICK: No. ACM is still a voluntary club. First of all it's not a professional society and secondly it's not active. Howard is suggesting some guys who get paid to work at these things, and not just whip off bright ideas.

GORDON: Several manufacturers have guys who are paid to work at these things.

PATRICK: We may be paying the wrong people. This is the advantage of an institute.

BROMBERG: This might be a good point. Lots of the people on the COBOL committee have additional, separate responsibilities. They do not put in full time at it.

PATRICK: This is somewhat like what SDC recommended here in TM-688; namely, that they house the institute and the government fund the thing. They speak of gathering a crew of experts together, not necessarily on SDC's payroll, to attack this very problem. That may be a relatively rational approach. If you separated the men from IBM and RCA and other manufacturers and put them in one place, they'd have to get along eventually--they can't pick up the phone and ask to come home because no one is listening to their ideas.

OPLER: This better be housed on one of the off-shore islands.

* : Christmas or Easter?

GORDON: If you'll pardon an impertinent question, what are the universities doing?

PATRICK: Bernie is experimenting and I don't think anyone else is doing anything else at all.

GRUENBERGER: At our 1959 symposium we ripped the universities apart and they haven't been heard from since. This is Bernie's second year and if he comes again next year we're going to have to give him the chair, gold plated.

GALLER: Universities have no authority in this field.

GORDON: You don't want authority. We already have too many guys with too little competence and too much authority. What I'd like to see is guys with more competence and no authority.

GALLER: Maintenance implies authority.

BROMBERG: For example, suppose you have an implementation problem in COBOL. How do you resolve it?

GORDON: I ask Bill Donally, who is the IBM representative on the COBOL committee, and he resolves it. He will give us an answer based on his discussions with the COBOL people, or on his knowledge of what was discussed when this thing came up. His answer is based on his best knowledge and he will eventually go back and check with the committee. If he is completely at a loss we will make a decision, based on implementation considerations. But by and large, Donally's full time job with no other responsibilities is to stay with the COBOL development and keep informed on what is intended

and we are guided by him.

PATRICK: He makes some of Ascher's 2^{nth} sub-decisions which make the languages not identical.

GORDON: But it's not random. We base it on the best information available to us.

PATRICK: But that's not like getting it from the horse's mouth.

GORDON: It's as close as we can get to the horse's mouth.

BROMBERG: I have great difficulty understanding how after 12 or 15 COBOL implementations are nearly completed that all of a sudden the maintenance group says, "I think now we ought to collect all these known ambiguities and start resolving them."

GORDON: Howard, you would have less difficulty understanding if you got back on the committee.

BROMBERG: No thanks. I'm just bringing these points up for the edification of those who have not been so fortunate as to attend one of these meetings.

OPLER: I hope I never see the day that we will resolve some of these questions by punching up a few cards and going down to the machine and feeding them in.

BROMBERG: Without such an authoritative organization it leaves the world open to perversions of interpretations of these common languages. Anyone can advertise, "Look, fellows, we now have a COBOL compiler."

BROMBERG: Complete with pictures, as a matter of fact. This certainly defeats what we're trying to do. So I put in a plea for a strong authoritative maintenance Czar with two bull whips.

GORDON: I don't think you should enforce something until you have something worth enforcing. With premature standardization, forcing it is compounding the error.

BROMBERG: No one has standardized.

McCRACKEN: Don't give him a bull whip until you're sure there's something in his skull.

PATRICK: They may go hand in hand, though. You can't give someone the responsibility without the authority.

GRUENBERGER: Are you saying that we should have stronger design control?

BROMBERG: Absolutely.

GORDON: No, you're not. You're saying, "Let's take what we're stuck with now and make the best of a bad situation by maintaining it stronger."

BROMBERG: What's involved in maintenance? There are three things: interpretation, modification, and extension. Modification may completely revise what we've got.

GORDON: Lots of luck. You know better than that.

BROMBERG: Why? It could, but it can't if every implementer goes his merry way and collects his own little ambiguities after the fact and presents them to the committee as accomplished facts. And what happens with the previous years'

specifications? Suppose now some slow guy comes running into the committee with a dozen and a half proposals for last year's COBOL specifications. You'd have 15 guys faint.

GRUENBERGER: Are all you guys saying that what we ought to do is to perfect a higher level language, set up a maintenance committee with a Czar and all that, and then worry about commonness?

BROMBERG: No, that in itself takes a giant step toward a guarantee of commonness.

GRUENBERGER: But you're also saying that we're not ready for commonness until those conditions are met, aren't you?

BROMBERG: I'm saying that we'll never have it until those conditions are met. I say we're ready and we're well overdue for it.

GRUENBERGER: Is there agreement on that in the group? That was the subject for discussion today. Are we saying that's a necessary and sufficient condition, before we even approach commonness?

CLIPPINGER: I wouldn't go along with that.

*: It's necessary but not sufficient. I think there's a lot of merit in what Howard proposes. But I don't think he was giving you an answer to the question you raised.

GRUENBERGER: Well, most of you indicated that you agreed it was a necessary condition but Dan, you didn't agree. Why not?

McCRACKEN: I'm not an implementer and I'd like to hear more about the conditions here. I'd still like to hear more about the idea of agreeing beforehand that we're going to try to maintain commonness and work within that framework. We should find the best people we can, set up the committee first, and then agree to stick with it.

GRUENBERGER: You mean get commonness first before we have a language that is maintained?

McCRACKEN: It's a question of states' rights. Are all the states going to have to perfect their own government before we can have a federal government?

BROMBERG: The original CODASYL charter said just this: there are two reasons why we're going to establish this necessary committee function for the creation of what turned out to be COBOL. One is the provisions of an effective tool for the specification of data processing problem solutions and the other is to maintain some degree of compatibility. That was absolutely stated before one little line was ever written down as far as the COBOL specifications were concerned.

OPLER: How common? When we talk about a common language do we mean that we expect to take the card decks of the COBOL Procedure Division and whomp them into the machine, or do we mean that we read the program and then start working? If you mean exact compatibility, where you take the COBOL deck, read it in a machine and answers come out, forget it. If you're saying that compatibility can be obtained at very low cost, then it makes some sense. The 200 decisions I was

talking about before will prevent you from ever putting the deck in any machine and getting correct answers.

McCRACKEN: All I want is something that's a lot better than what we have now. It doesn't have to be perfect.

ARMER: Where are we going to be two years from now? Are you saying that if I'm a guy who uses one manufacturer's machine that I'm in a sense, no longer tied to him? Not that I can pick up my cards and run them in this new machine but that the cost of changing over is not as great as it is today? That maybe there is some transference of training among my fairly talented programming staff?

McCRACKEN: That's precisely what I'm saying.

ARMER: Isn't that a big improvement over the state of things we have today?

McCRACKEN: It'll cost me $50,000 to change machines instead of half a million dollars and I'd say that's a lot better.

GRUENBERGER: Even though we pay a terrific price to get that conversion reduction down to $50,000?

GORDON: That's right, is it a matter of saving the $450,000 or is it a matter of paying it out in a different form, over a longer period?

* : I agree, look at that equation of Patrick's.

ARMER: If we look at these goals instead of the goals that the common man in the data processing business thinks these goals are, then maybe we can do it a lot cheaper.

McCRACKEN: I'm just thinking about the guy that had to get himself machine B that he desperately didn't want. He's paying for it. He's paying machine rental every month without getting the compatibility.

GRUENBERGER: I was just looking at Patrick's equation there. Supposing it costs you 10 cents more for every machine run you make. And every three years you save nearly half a million dollars in conversion costs. You come out about the same. It's just bookkeeping.

McCRACKEN: I'd love to be able to have the choice.

GRUENBERGER: That would help, if you could make it freely. I don't think you can.

DOBRUSKY: Standards and commonness are going to cost you at some level, I don't care where you put it.

McCRACKEN: Of course.

DOBRUSKY: It's dependent upon these trade-offs and the long run of it in training. Perhaps it's completely intangible because we don't even recognize it.

GRUENBERGER: We've noticed that in industry there are certain costs that you can bury very neatly. And those that you can bury you don't see any more.

LITTLE: You can sit and talk about this all day, but tomorrow somebody starts a new job. Two years from now we'll have to face the fact that we have been using some of these new languages over the last two years, and if we aren't in better shape then, we've been kidding ourselves.

GRUENBERGER: At least most of our present programmers will have two more years of experience.

LITTLE: Yes, but they will all be supervisors someplace.

* : They'll be compiler writers!

BROMBERG: I think we will be in substantially better shape. In a practicable fashion people will be able to emancipate themselves from this slave market business with their present computer vendor. People will be able to look at the entire market.

GRUENBERGER: Barry, would you make that same speech?

GORDON: As far as emancipation is concerned I would like to point out that the electronic emancipation proclamation was written by IBM several years ago and it is called FORTRAN. Philco and Honeywell and CDC and a great many other people have made excellent use of it. Over the years it has proven itself to be quite valuable, for IBM, for IBM customers, for former IBM customers, for our competitors, and for the whole industry. Now, five years later, it is being seriously considered as a standard. In another half decade (or less, since we're getting smarter faster) we may be able to do the same thing in the commercial area. Maybe we can even do it in the Command and Control area and in other areas. I think you should first find out what you're doing before you legislate it into a standard.

ARMER: Is it not true in the Command and Control area that we have had experience with a few of these languages

but at the time that somebody froze on COBOL there had been essentially no experience?

GORDON: Essentially that is correct. At the time that COBOL was chosen, there had been very little experience.

* : There'd been a fair amount of experience among FLOWMATIC users.

GORDON: FLOWMATIC is not one of these languages, as you well know.

McCRACKEN: One of what languages?

GORDON: The so-called higher level English-type narrative languages. FLOWMATIC is about at the Autocoder-2 macro-instruction level.

McCRACKEN: That is not my impression at all.

GORDON: You can hold up the opinion bag but I'll stand by it. FLOWMATIC is just not in the same ballpark.

* : It looks like it but it's not.

PATRICK: That's right. FLOWMATIC is very weak and AIMACO wasn't working when the COBOL effort went into orbit.

OPLER: A few weeks ago I was sitting with Dick Clippinger and we were trying to come up with a few figures to estimate what percentage of data processing problems now running in this country have been coded with the help of a data processing compiler. It's only a guess, but we concluded it was about one per cent. Our next question was, what percentage of the programs now being written in data processing are using a data processing compiler. Again it's just a guess but we

estimated that it was less than 10 per cent. It would be interesting to see how these figures change in the next few years.

ARMER: Is it not apt to happen, particularly with respect to the small machines, that we'll find people abandoning these narrative languages and going back to assembly programs?

PATRICK: Because the machines are too small to compile on.

GRUENBERGER: Mr. Patrick here has devised an interesting test for compilers. He asks a simple minded question,"How long would it take this compiler to compile a HALT?" We've got some figures on this, and they're very interesting.

BROMBERG: That sounds awfully impractical. Who wants to compile a HALT?

GRUENBERGER: We do, as a matter of fact.

BROMBERG: Well, we'll build it into our compiler. If you want to do it, it'll be a special case. We'll do it in 30 seconds.

GRUENBERGER: Yes, and that's exactly what happens. But just as a matter of fact (this is pertinent to what you just said, Ascher) we get times like 20 seconds in FORTRAN, 20 seconds in JOVIAL, 90 minutes in FORTRAN on the tape 1620.

ARMER: That's a fact?

GRUENBERGER: I may be exaggerating that last one a little bit; maybe it isn't 90 minutes, maybe it's only an hour.

McCRACKEN: What is it in MAD, Bernie?

GALLER: On the 7090, if you subtract all the time necessary to load, it's about 1 second.

OPLER: And that's the whole program?

KOORY: You're talking about the total job time?

GALLER: If you want to throw everything else in the total job time, it would be perhaps 5 seconds.

GRUENBERGER: When you go to small machines the threshold effect can kill you. In order to get the system chunking away so that you can compile anything, you have to chew up a fair amount of time.

MacKENZIE: You're just checking the algorithms that were used in implementation, not the commonness of the language itself.

McCRACKEN: But these are related.

ARMER: If you're talking about having a common language for small machines, when you can show that a common language for small machines is no good, then you're crazy.

GALLER: It's not the language, it's the translator.

BROMBERG: You're talking now about the overhead time that any system requires to do a certain amount of work. It just happens to be very high for a small machine.

PATRICK: Right, that's the intersect.

ARMER: I'd like to get more out of Howard about my statement that with small machines there's going to be a great movement back to assembly programs from the magic languages. Nobody seemed to argue out loud but you seemed

to indicate that you would.

BROMBERG: I think that's wrong for two reasons. First, I would imagine that the type of individual who is working on these small machines would find it necessary, for their livelihood, to deal with these languages.

McCRACKEN: What does that mean?

BROMBERG: They're not programmers. They're tab people. These are the people who, when the 1401 came out, ran down the hall and put in an order for 6 of them. Now, they've got to use them. One way to get to use them is through one of these languages.

ARMER: And now all he's doing on his machine is compiling.

BROMBERG: Probably what he's doing is using his machine as a glorified tabulator.

GORDON: We have utility programs that do that without having to compile.

GRUENBERGER: Or generators.

BROMBERG: Well, you guys are going to have a COBOL processor for the 1401 aren't you?

PATRICK: Because the DOD said they had to have one.

GORDON: There is an announcement to that effect.

McCRACKEN: Let's move up to the 1410.

BROMBERG: All right, let's.

ARMER: A COBOL translator, I understand, exists for the 1410. Are we going to be throwing it out in six months?

McCRACKEN: That isn't the whole story. I know of a 501

installation that does their coding in absolute octal. It isn't just a question of availability, or how good it is; there are a lot of other factors. They had the COBOL course.

OPLER: We know of one of the earliest cases of disillusionment of a user of a medium sized machine using a COBOL processor on a job that could be done easily in assembly language.

GALLER: We ought to keep clear the distinction between the language and the translator. We're judging languages in terms of the current translators, which is very unfortunate.

OPLER: Let me go further on the subject. We just completed the design of a COBOL translator for a small machine. The size and scope of the COBOL language is such that by no stretch of the imagination is there any way to fit it in the core of the machine; therefore, it must be done by multiple passes, and takes relatively a long time to compile. Therefore, I am stating that the COBOL language, itself, is such that on a machine with a small core memory and limited tape ability, it will take a long time to compile <u>any</u> COBOL program.

GALLER: Did you, for example, organize it so that most of the features that a person might use could be done by a small part of the translator, with the rest brought in only if the user gets exotic?

OPLER: It turns out that this is very difficult if at all possible.

GORDON: Yes, for example, a feature that a programmer might use is data description.

OPLER: In a standard method of using COBOL, a large corporation will take all of its master files and make huge data descriptions for them. Now, a COBOL program, to be compiled on their computer, may be of the form "Open file, read in a record, determine whether this employee is over 21 or not, close the file." To process this four-statement procedure, you have to bring the entire data description of this tremendous file into the peewee memory of the computer.

BROMBERG: There are certain techniques... On a small machine one could consider doing an initial compilation, keeping in mind a finite number of files that you're going to pass through the machine in this installation. You create a mechanism wherein the processed data description of that particular file as it exists now during the initial compilation in the internal manipulatible compiler language is library stored, and you save a lot of compile time.

OPLER: But you can't convince a customer that this is the way he has to operate.

GORDON: The original definition of COBOL specifications says that the library must contain material in source language form. This is what you call an implementer's unique original extension. And it's very common.

GORDON: You still don't have the COBOL specifications. Either you're going to talk effective or you're going to talk common.

BROMBERG: You put in a library the original source language according to the DOD specifications and your internal

language. The function of the internal language is naturally for fast compilation. The function of the original source language is for the reference format.

OPLER: In other words, here is COBOL, and here is the way to get around it.

BROMBERG: I'm just worrying about fast compilation for the problem Armer brought up about the small machines. It doesn't violate anything. It obeys all the rules of COBOL.

PATRICK: But if you have only 32,000 words and the translator itself takes about 70,000 it seems to me it's going to be like multi-pass.

BROMBERG: I'm not going to give out any more tips to you guys.

PATRICK: There's one way around it: you can put a whooping big drum over in the power supply and not tell the customer about it.

MacKENZIE: I don't think that's quite fair, Bob. Please don't rule out the fact that there would probably be more worthwhile reasons for having that drum in the system. For example, it might be there for organizational reasons not related to its ability to process some particular language.

PATRICK: If I have a language of reduced sophistication and the whole translator takes only 20,000 instructions there is a good chance it will be a one-pass compiler.

OPLER: I'd appreciate some of this advice with regard to a 60,000-word compiler that is to operate on a 2000-word machine.

GORDON: Does he want to write out a c
advice?

PATRICK: If there are no further pro
I think we stand adjourned.

GORDON: Would you like to hold up th
how much we accumulated in each one?

PATRICK: I can hardly lift the opini

OPLER: Howard, I really hope you can
laugh a couple of years from now.

DOBRUSKY: I've got a topic for next
"Measures of Compiler Language and System

GORDON: Make it "Measures of Program

(The meeting adjourned.)